Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Krašna

Zajem in analiza poti s sodobnimi tehnologijami in senzorji mobilne naprave

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Mira Trebar

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Funkcionalnosti mobilnih naprav, kot so pametni telefoni, so vse bolj povezane z vgrajenimi senzorji in drugimi tehnologijami. V diplomskem delu naj kandidat razvije mobilno aplikacijo, ki zajema in analizira podatke navigacijskega sistema, vremenskega vmesnika, barometra, števca korakov za različne aktivnosti in tehnologije NFC. Rezultate naj predstavi z realnimi primeri opravljene poti na zemljevidu, v grafih in statističnih izpisih na zaslonu ali v datoteki za kasnejši pregled.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application Programming Interface	vmesnik za programiranje aplikaciji
GDOP	Geometric Dilution of Precision	geometrijska razredčitev natančnosti
GPS	Global Positioning System	globalni sistem pozicioniranja
GPX	Global Positioning System Exchange Format	oblika izmenjave sistema globalnega pozicioniranja
HTTPS	Secure Hypertext Transfer Protocol	zavarovana različica prenosnega protokola
IDE	Integrated Development Environment	integrirano razvojno okolje
JSON	JavaScript Object Notation	objektna notacija JavaScript
LEO	Low Earth orbit	nizka zemeljska orbita
MEMS	Micro-Electro-Mechanical Systems	mikro-elektronsko-mehanski sistemi
NDEF	NFC Data Exchange Format	format izmenjave podatkov NFC
NFC	Near Field Communication	komunikacija kratkega dosega
RFID	Radio Frequency identification	radiofrekvenčna identifikacija
XML	Extensible Markup Language	razširljiv označevalni jezik

Povzetek

Naslov: Zajem in analiza poti s sodobnimi tehnologijami in senzorji mobilne naprave

Avtor: Rok Krašna

Sodobne mobilne naprave, predvsem pametni telefoni so opremljeni s senzorji, ki razširjajo njihovo funkcionalnost. Cilj diplomske naloge je izdelava prototipa mobilne aplikacije za pametne telefone, ki z uporabo senzorjev in drugih tehnologij beleži podatke in na njihovi osnovi izvede analizo poti. Razvoj aplikacije združuje uporabo sistema GPS (angl. Global Positioning System), vmesnika OpenWeatherMap, senzorskih tehnologij, kot sta barometer, števec korakov, in tehnologije za komunikacijo kratkega dosega (angl. Near Field Communication). Osrednji del opisuje delovanje in implementacijo algoritmov, ki skrbijo za pridobivanje, obdelavo in shranjevanje podatkov, poleg tega pa so podani tudi funkcionalnosti zajema poti na zemljevidu in statistični rezultati. V zaključku so predstavljeni postopki in rezultati testiranja ter možnosti za nadgradnjo in nadaljni razvoj aplikacije.

Ključne besede: senzorji, pameti telefon, Android, GPS, barometer, NFC

Abstract

Title: Route capture and analysis using modern technologies and mobile device sensors

Author: Rok Krašna

Modern mobile devices, especially smartphones, are equipped with various sensors that extend their functionality. The goal of the thesis is to create a prototype of mobile application for smartphones, which uses sensors and other technologies to record the data and performs a path analysis on their basis. Application development combines the use of the Global Positioning System, the OpenWeatherMap weather interface, sensor technologies such as barometer, sensor for counting the number of steps and Near Field Communication technology. The central part describes the operation and implementation of algorithms that take care of the acquisition, processing and storage of data, as well as the functionality of capturing paths on the map and statistical results. The conclusion shows the procedures that were used for testing and test results as well as the possibilities for improvements and further development of the application.

Keywords: sensors, smartphone, Android, GPS, barometer, NFC

Poglavje 1

Uvod

Mobilne naprave, predvsem pametni telefoni so v sodobnem svetu za marsikoga nepogrešljiv spremljevalec. Po podatkih iz leta 2017 je na svetu 4,7 miljarde mobilnih naprav, njihovo število pa se iz leta v leto povečuje. Prav tako se povečuje tudi zmogljivost njihovih komponent in s tem možnosti uporabe. Mobilne naprave že nekaj let niso več namenjene izključno za telefoniranje in pošiljanje sporočil. Omogočajo dostop do svetovnega spleta, zajem fotografij in videa visokih ločljivosti, navigacijo GPS in mnogo drugih funkcionalnosti, ki se jih uporabniki pogosto spoh ne zavedajo. V to kategorijo spadajo vgrajeni senzorji, ki razvijalcem ponujajo zanimive možnosti za razvoj mobilnih aplikacij. Eno od področji, kjer lahko uporabimo senzorje mobilne naprave v povezavi s tehnologijo GPS, je spremljanje poti uporabnika med izvajanjem različnih aktivnosti.
V diplomski nalogi je predstavljen prototip mobilne aplikacije, ki z uporabo GPS, vgrajenih senzorjev in algoritmov spremlja uporabnika med vožnjo, tekom in hojo. Aplikacija beleži podatke o opravljeni razdalji, času, hitrosti, nadmorski višini, smeri potovanja, temperaturi in jih po končanem spremljanju shrani v datoteko. Uporabnik lahko v realnem času spremlja svoje aktivnosti. Aplikacija na osnovi shranjenih poti prikaže zgodovino gibanja uporabnika in odstotek izvajanja posameznih aktivnosti. Za vsako shranjeno pot omogoča tudi izvedbo analize in prikaz rezultatov na zemljevidu in grafih.

Poglavje 2

Pregled področja

V tem poglavju bodo predstavljene tehnologije, ki so bile uporabljene med razvojem aplikacije. Opisali bomo delovanje sistema GPS in dejavnike, ki vplivajo na natančnost izračuna lokacije. V nadaljevanju si bomo pogledali senzorske tehnologije, vmesnik za vremenske podatke in razvojno okolje Android Studio.

2.1 GPS

Globalni sistem pozicioniranja ali GPS [9] je tehnologija, ki omogoča pozicioniranje naprav z ustreznim sprejemnikom. Sistem je bil prvotno razvit za potrebe ameriške vojske, po letu 1980 pa je na voljo za civilno uporabo. Sprejemniki so integrirani v širok spekter mobilnih naprav, kot so na primer pametni telefoni, tablični računalniki, navigacijski sistemi.

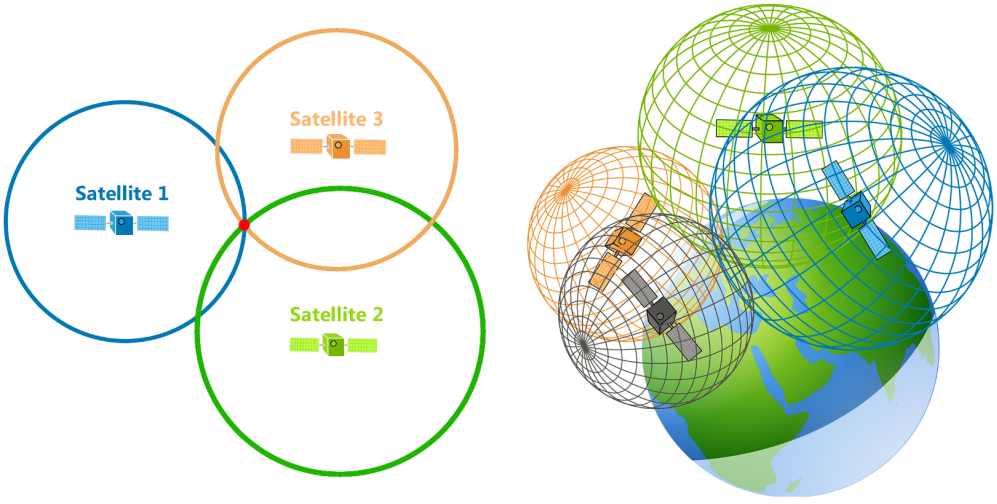
2.1.1 Delovanje GPS

Sistem GPS je zgrajen iz konstelacije 31 satelitov (24 jih je vedno aktivnih). Sateliti krožijo v srednji zemeljski orbiti na nadmorski višini približno 20.200 km in dnevno dvakrat obkrožijo Zemljo. Razporejeni so v šest orbitalnih ravnin. Takšna postavitev zagotavlja, da sprejemniki vidijo najmanj štiri satelite s skoraj vsake točke na Zemlji. Sateliti oddajajo krožne polarizirane

signale na dveh frekvencah, označenih z L1 in L2 [23]. Glavni nosilec signala je frekvenca L1 na 1575 MHz, ki je modulirana z dvema kodama:

- `C/A` ali koda grobe pridobitve (angl. coarse/acquisition), znana tudi kot civilna koda. To je koda, ki jo uporabljajo sprejemniki v večini pametnih telefonov.
- `P/Y` ali koda natančnosti in varnosti (angl. precision/secure), ki je kriptografsko zaklenjena za potrebe vojske in pooblaščenih oseb.

Frekvenca L2 na 1227 MHz je modulirana samo s kodo natančnosti `P` in se uporablja za korekcijo napake satelitskega signala, ki nastane zaradi refleksije v atmosferi (poglavje 2.1.2). V razvoju sta tudi frekvenci L1C in L2C ter frekvenca L5, ki je namenjena visoko zmogljivim napravam. Za izračun lokacije naprave sprejemniki uporabljajo princip trilateracije (Slika 2.1).



Slika 2.1: Določanje lokacije na površju Zemlje po principu trilateracije [19].

Princip trilateracije

Vsi sateliti oddajajo signale, ki jih sprejemnik GPS dekodira ter tako pridobi časovno značko in oddaljenost satelita. Za uporabo trilateracije [19] mora

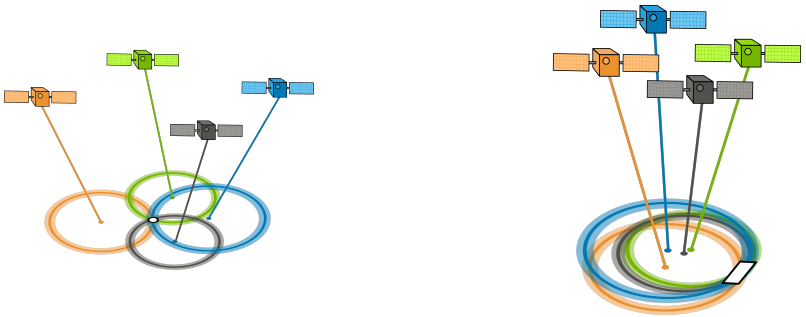
sprejemnik pridobiti podatke najmanj štirih satelitov. Satelit se nahaja v središču sfere, presečišče sfer pa označuje lokacijo sprejemnika.

2.1.2 Natančnost lokacije GPS

Natančnost, s katero GPS naprave določajo lokacijo, je odvisna predvsem od zmogljivosti sprejemnika. Sprejemniki GPS v večini sodobnih mobilnih naprav dosegajo natančnost do petih metrov v 95 % primerov [14], medtem ko zmogljivejši sprejemniki dosegajo bistveno boljše rezultate. Na natančnost vplivajo tudi nekateri zunanji dejavniki. Najpogostejši vzroki za napake so refleksija v atmosferi, razporeditev satelitov GDOP (angl. Geometric Dilution of Precision), večtočkovni učinek (angl. Multipath) in kvaliteta sprejemnika GPS.

Napake zaradi razporeditve satelitov

Vrednost GDOP (angl. Geometric Dilution of Precision) predstavlja napako sprejemnika GPS, ki jo povzroča relativni položaj satelitov [13]. Položaj vpliva na število signalov, ki jih sprejme sprejemnik. Kadar je položaj satelitov bolj razširjen (Slika 2.2a), ima sprejemnik dobro vrednost GDOP, če pa so sateliti bližje drug drugemu (Slika 2.2b), je vrednost GDOP slabša. Takšna napaka zmanjša natančnost lokacije za nekaj metrov.

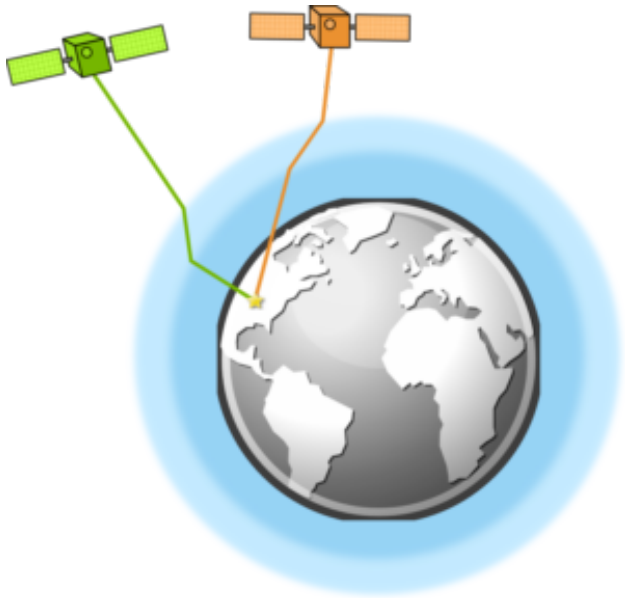


(a) Dobra razporeditev satelitov. (b) Slaba razporeditev satelitov.

Slika 2.2: Razporeditev satelitov v primeru dobre in slabe vrednosti GDOP.

Napake zaradi refleksije v atmosferi

Troposfera in ionosfera vplivata na hitrost širjenja signala satelitov. Zaradi razmer v atmosferi in spremembe gostote zraka pride do spremembe smeri potovanja satelitskih signalov pri njihovem prehodu na zemeljsko površje. Večina sprejemnikov GPS prejema satelitske podatke samo preko frekvence `L1` (poglavje 2.1.1), kar še dodatno poveča možnost za napake. Glede na razmere v atmosferi lahko ta vrsta napake povzroči odstopanja, ki so lahko večja od petih metrov. Slika 2.3 prikazuje vpliv atmosfere na širjenje signala GPS.

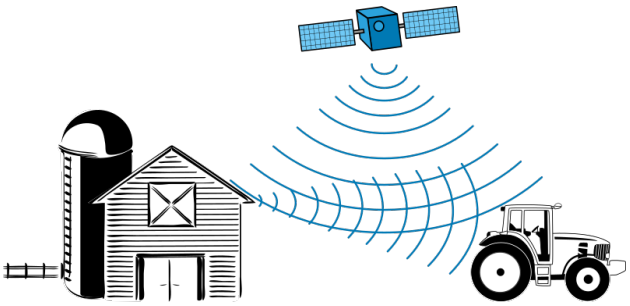


Slika 2.3: Vpliv atmosfere na širjenje satelitskega signala pri prehodu na zemeljsko površje.

Problem refleksije v atmosferi [13] rešujejo dvoplastni sprejemniki GPS in razširitveni sistemi. Takšni sprejemniki uporabljajo podatke, ki jih sateliti posredujejo preko frekvence `L2`, kar omogoča natančnejši izračun lokacije. Dvoplastni sprejemniki omogočajo pozicioniranje v realnem času z natančnostjo v centimetrih, vendar so bistveno dražji in jih zato najdemo v napravah za zahtevnejše uporabnike.

Napake zaradi večtočkovnega učinka

Interferenca, ki jo povzroča večtočkovni učinek (angl. Multipath), [13] nastane takrat, ko se satelitski signal odbije od bližnjih objektov (Slika 2.4), kot so stavbe, mostovi, gore ali drevesa. Večtočkovni učinek ima večji vpliv na satelite LEO (angl. Low Earth Orbit), ki se nahajajo v nizki zemeljski orbiti (2.000 km nad površjem Zemlje). Napaka, ki nastane, se razlikuje glede na frekvenco vendar redko dosega odstopanja, ki so večja od treh metrov.



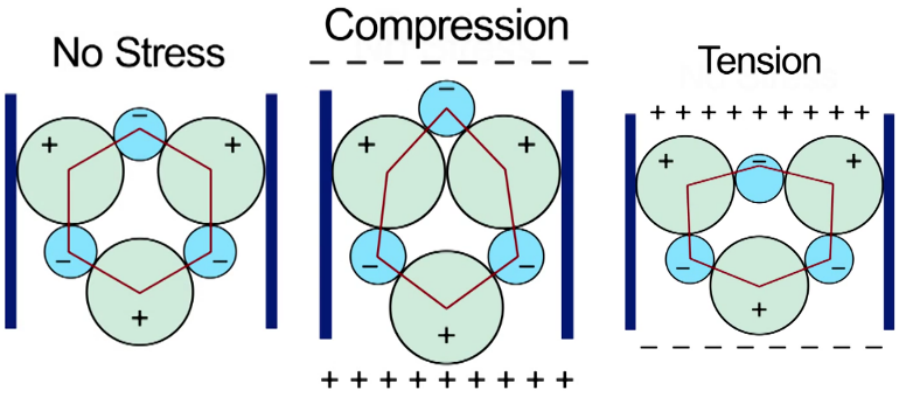
Slika 2.4: Nastanek večtočkovnega učinka pri odboju satelitskega signala od stavbe.

2.2 Barometer

Barometer je naprava za merjenje zračnega tlaka. Mobilne naprave namesto skalnih barometrov uporabljajo senzorje MEMS (angl. Mico-Electro-Mechanical System) [21]. Večina tlačnih senzorjev v mobilnih napravah spada v kategorijo piezo-rezistivnih (angl. Piezo-Resistive) senzorjev. Z barometrom lahko ugotovimo višino naprave in vremenske spremembe ter izboljšamo natančnost GPS. Uporablja jih širok spekter naprav, med katere spadajo prenosne navigacijske naprave, pametni telefoni in zunanji števci korakov.

Delovanje piezo-rezistivnega tlačnega senzorja

Piezo-rezistivni senzor tlaka vsebuje več tankih plasti silicija, ki so vgrajene med zaščitno membrano [3]. Za merjenje zračnega tlaka izkorišča mehanske lastnosti silicija, ki se pod stresom upogne, kar povzroči spremembo v strukturi piezo kristalov (Slika 2.5). Sprememba strukture vpliva na električno upornost materiala. Senzor na podlagi spremembe upornosti izračuna vrednost zračnega tlaka v milibarih.



Slika 2.5: Sprememba strukture piezo kristalov pod stresom.

2.3 Števec korakov

Senzor za zaznavanje in štetje korakov [4] spada v kategorijo strojnh senzorjev. Vgrajen je v večino sodobnih pametnih telefonov. Deluje na osnovi pospeškometra, ki meri gravitacijsko silo glede na napravo. Omogoča štetje korakov med tekom, hojo in gibanjem po stopnicah. Senzor je dovolj pameten, da prepozna aktivnosti, kot so kolesarjenje, vožnja z avtomobilom ali vlakom, in v takšnih primerih ne izvaja šteja korakov. Prvi senzor je `Step Detector`, ki po vsakem zaznanem koraku vrne vrednost ena, kar pomeni, da mora aplikacija sama poskrbeti za štetje. Prednost `Step Detectorja` so zelo majhne zakasnitve, kar pomeni, da je čas spremembe števca tesno povezan

s koraki uporabnika. Drugi senzor je `Step Counter`, ki tako kot prej omenjeni senzor vrača vrednost po vsakem zaznanem koraku. Razlika je v tem, da `Step Counter` sam poskrbi za prištevanje in vrne skupno število korakov od začetka štetja. V primerjavi s `Step Detectorjem` ima nekoliko večjo zakasnitev med štetjem, vendar proizvede manjše število `False/Positive`[1] vrednosti, zato je primernejši, kadar je potrebna večja natančnost.

2.4 OpenWeatherMap API

OpenWeatherMap [17] je spletni vmesnik, ki razvijalcem spletnih in mobilnih aplikacij omogoča dostop do vremenskih podatkov, ki so na voljo v treh različnih formatih. Za vir podatkov uporablja meteorološke oddajne storitve ter neobdelane podatke iz letaliških, radarskih in drugih uradnih vremenskih postaj. Za uporabo API-ja je treba ustvariti račun, kjer lahko generiramo unikaten ključ API (angl. Application Programming Interface). Osnovna storitev posodablja podatke v približno dvournih intervalih, je brezplačna in omogoča do 60 zahtev na minuto. Za pridobitev natančnejših podatkov in večje število zahtev se storitev ustrezno obračuna v obliki mesečne naročnine.

2.5 NFC

NFC (angl. Near Field Communication) [15] je tehnologija, ki omogoča komunikacijo kratkega dosega med združljivimi napravami. Komunikacija zahteva vsaj eno napravo, ki oddaja signal, in napravo, ki sprejema signal. Naprave delimo na:

- `Pasivne` – v to kategorijo spadajo različne značke (angl. tag) in drugi majhni oddajniki, ki lahko pošljejo informacije drugim napravam. Pasivne naprave ne obdelujejo nobenih informacij, ki jih prejmejo od dru-

[1] False/Positive: sprememba gibanja, ki jo senzor zazna kot korak, čeprav korak ni bil opravljen.

gih naprav, in se ne morejo povezati z drugimi pasivnimi komponentami.

- `Aktivne` – te naprave lahko pošiljajo ali prejemajo podatke in komunicirajo med seboj ali z drugimi pasivnimi napravami. Pametni telefoni so daleč najpogostejša oblika aktivne naprave NFC.

Delovanje NFC

Tako kot Bluetooth, Wi-Fi in druge brezžične tehnologije NFC deluje po principu pošiljanja informacij preko radijskih valov. Tehnologija, ki se uporablja v NFC, temelji na poznani tehnologiji RFID (angl. Radio Frequency identification), ki za prenos informacij uporablja elektromagnetno indukcijo. To je tudi bistvena razlika med NFC in Bluetooth ali Wi-Fi, saj pasivne naprave ne potrebujejo svojega vira energije za napajanje. Napaja jih elektromagnetno polje, ki ga proizvaja aktivna komponenta, ko pride v njihov doseg.

2.6 Razvojno okolje Android Studio

Android Studio IDE (angl. Integrated Development Environment) je razvojno okolje, ki je prilagojeno za razvoj aplikacije na Googlovem operacijskem sistemu Android. Razvila ga je ekipa Googlovih inženirjev na osnovi priljubljenega razvojnega okolja IntelliJ IDEA. Android Studio je bil prvič predstavljen v razvojni fazi na konferenci Google I/O 2013. Razvijalcem je uradno postal na voljo decembra 2014, ko je bila objavljena prva stabilna verzija 1.0 [20]. Zaradi številnih prednosti in izboljšav je nadomestil prejšnje razvojno okolje Eclipse z dodatkom ADT. Android Studio ima podporo na operacijskih sistemih Windows, Linux in Mac.

Poglavje 3

Razvoj prototipa aplikacije

V nadaljevanju si bomo pogledali razvoj in funkcionalnosti, ki jih ponuja aplikacija, in kakšne so bile odločitve pri izbiri mobilne platforme za razvoj. Predstavljeni bodo zgradba aplikacije, načini pridobivanja podatkov in implementacija pomembnejših algoritmov. Zaključek poglavja je namenjen posameznim aktivnostim in njihovemu delovanju ter izgledu uporabniškega vmesnika.

3.1 Definicija funkcionalnosti in zahtev

Med glavne funkcionalnosti spadajo beleženje podatkov, izvedba in prikaz rezultatov analize poti na osnovi podatkov, ki jih aplikacija pridobi z uporabo senzorojev ter vmesnika OpenWeatherMap. Za izvedbo analize je potreben dostop do podatkov barometra, števca korakov (pri teku in hoji), navigacijskega sistema GPS, za uporabo vremenskega vmesnika pa je potrebna internetna povezava. Prikaz rezultatov analize vključuje splošni pregled parametrov, kot so skupna dolžina, porabljeni čas, povprečna in največja hitrost, podatki o nadmorski višini in vremenu. Drugi del izriše pot na zemljevidu in omogoča prikaz hitrosti, nadmorske višine, vzponov in spustov na posameznih odsekih. Dodatne funkcionalnosti vključujejo pregled zgodovine, statistike, izvoz shranjenih poti in zapis značke NFC za zagon aplikacije.

Zgodovina vsebuje seznam vseh shranjenih poti in prikaz njihove analize. Z uporabo statistične analize so izdelani mesečni prikaz opravljenih razdalj, število korakov, porabljene kalorije in odstotek aktivnosti. Prikaz zgodovine in statistike lahko uporabnik omeji na določeno časovno obdobje z uporabo filtra za prikaz vsebine. Izvoz shranjene poti je na voljo v dveh formatih. Namen izvoza je ustvarjanje kopije poti v primeru izgube podatkov na mobilni napravi in možnost uvoza v druga programska orodja, kot so Google Earth Pro, in uporaba v drugih aplikacijah. Zapis značke NFC omogoča zagon aplikacije, kadar napravo postavimo v njeno neposredno bližino. Funkcionalnost podpira vse pasivne značke, ki omogočajo shranjevanje sporočila NDEF (angl. NFC Data Exchange Format).

3.2 Izbira mobilne platforme

Za razvoj aplikacije smo izbrali odprtokodni operacijski sistem Android, ki temelji na Linux jedru. Poganja večino pametnih telefonov in mnogo drugih naprav, za njegov razvoj pa od leta 2005 skrbi podjetje Google. Razlogi za njegovo izbiro so naslednji:

- Najbolj razširjen mobilni operacijski sistem, kar pomeni večje število potencialnih končnih uporabnikov.
- Dobra dokumentacija [1], prakse ter velika skupnost razvijalcev, ki pripomorejo k lažjemu iskanju pomoči med razvojem.
- Širok izbor zunanjih knjižnic in API-jev za hitrejši razvoj.
- Ni potrebe po učenju novih programskih jezikov, ker razvoj poteka v že poznanem programskem jeziku Java.

V okviru operacijskega sistema smo se odločili za različico 7.0, imenovano "Nougat", ki je s 23,3 % tržnim deležem med razvojem aplikacije (Slika 3.1) tretja najpogostejša različica Androida. Pri odločitvi smo upoštevali dejstvo,

da se je njen tržni delež v letu 2017 podvojil in da z rastjo nadaljuje v letu 2018.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.5%
4.1.x	Jelly Bean	16	2.0%
4.2.x		17	3.0%
4.3		18	0.9%
4.4	KitKat	19	13.4%
5.0	Lollipop	21	6.1%
5.1		22	20.2%
6.0	Marshmallow	23	29.7%
7.0	Nougat	24	19.3%
7.1		25	4.0%
8.0	Oreo	26	0.5%

Slika 3.1: Tržni deleži operacijskih sistemov Android [22].

3.3 Mobilna aplikacija

Aplikacija je sestavljena iz petih aktivnosti. Aktivnost (angl. Activity) je eden od temeljnih gradnikov aplikacij na platformi Android. Služi kot vstopna točka za interakcijo med uporabnikom in aplikacijo ter igra pomembno vlogo pri določaju, kako se uporabnik orientira znotraj aplikacije. Aktivnost lahko vsebuje eno ali več funkcionalnosti. Težava pri aktivnostih z več funkcionalnostmi je smiselen prikaz elementov uporabniškega vmesnika na zaslonu naprave. V takšnih primerih smo posamezne funkcionalnosti razdelili na več manjših modulov, ki smo jih implementirali z uporabo fragmentov. Fragment deluje na podoben način kot aktivnost. Ima svoj življenjski cikel, vhodne podatke in gradnike uporabniškega vmesnika, vendar izven aktivno-

sti ne more samostojno obstajati. Na vsako aktivnost in fragment je vezana vsaj ena datoteka XML (angl. Extensible Markup Language) za definicijo izgleda uporabniškega vmesnika. Datoteka vsebuje posamezne gradnike in parametre, ki določajo njihov izgled in postavitev na zaslonu.

Aktivnosti:

- **MainMenuActivity** – aktivnost z glavnim menijem in povezavami do ostalih aktivnosti;
- **CaptureActivity** – beleži podatke med izvajanjem poti;
- **DisplayDataActivity** – prikaže rezultate analize poti;
- **SettingsActivity** – prikazuje nastavitve aplikacije in omogoča njihovo spreminjanje;
- **NfcActivity** – skrbi za zapis značke NFC.

Fragmenti:

- **NewPathFragment** – pridobi podatke za pripravo nove poti;
- **HistoryFragment** – prikaže zgodovino shranjenih poti;
- **StatisticsFragment** – prikaže statistike shranjenih poti;
- **ExportFragment** – omogoča izvoz shranjene poti;
- **OverviewFragment** – prikazuje osnovne podatke poti;
- **MapFragment** – izriše pot na zemljevidu;
- **GraphFragment** – prikaže določene podatke poti na grafu.

3.4 Pridobivanje podatkov

3.4.1 Vremenski podatki

Za pridobivanje vremenskih podatkov je bila implementirana asinhronska operacija z uporabo razreda `AsyncTask`. Takšen način implementacije je primeren za izvedbo krajših operacij v ozadju. Podatke posreduje OpenWeatherMap API na osnovi zahteve HTTPS (angl. Secure HyperText Trasfer Protocol). Najprej je treba ustvariti URL, v katerem so definirane geografske koordinate lokacije in unikatni ključ API. V naslednjem koraku se izvede klic metode `openConnection()`, ki odpre povezavo HTTPS in pošlje zahtevo. Po obdelavi zahteve strežnik pošlje odgovor v formatu JSON (angl. JavaScript Object Notation), ki ga preberemo z objektom `BufferedReader` v povezavi z vhodnim tokom `InputStream`. Prebrana vsebina se shrani v objekt tipa `JSONObject`, ki omogoča dostop do dejanskih vrednosti vremenskih podatkov. Operacija nato prekine povezavo s klicem metode `disconnect()` in vrne seznam podatkov.

3.4.2 Podatki senzorjev in sistema GPS

Pridobivanje podatkov s senzorji mobilne naprave in sistema GPS smo implementirali z uporabo storitev (angl. Service). Storitev je komponenta aplikacije, ki omogoča izvajanje dolgotrajnih operaciji v ozadju in posredovanje vmesnih rezultatov za prikaz na uporabniškem vmesniku. Takšen način implementacije preprečuje težave z odzivnostjo uporabniškega vmesnika. Storitev je tako kot aktivnost treba deklarirati (Izsek 3.1) v datoteki `AndroidManifest`.

```
1 <service
2   android:name=".services.GpsSensorService"
3   android:exported="false"
4   android:label="GpsLocationService"
5   android:stopWithTask="true" />
```

Izsek 3.1: Deklaracija storitve v datoteki `AndroidManifest.xml`.

Prenos podatkov iz storitve v aktivnost smo implementirali po tako imenovanem `publish-subscribe` vzorcu. Storitev zapakira podatke v objekt za prenos sporočil `Intent` (Izsek 3.2). Vsak paket vsebuje objekt `IntentFilter` za identifikacijo paketa pri ustreznem sprejemniku. Podatki v sporočilu so zapisani v obliki parov ključ/vrednost in so lahko različnih podatkovnih tipov. Ko storitev pošlje paket, ga sistem samodejno usmerja do aktivnosti v aplikaciji, ki so se naročile na prejemanje te vrste paketov.

```
1 IntentFilter filter = new IntentFilter("current_speed");
2 Intent intent = new Intent(filter);
3 intent.putExtra("speed", location.getSpeed() * 3.6);
4 sendBroadcast(intent);
```

Izsek 3.2: Priprava in posredovanje paketa, ki vsebuje podatek o hitrosti.

Aktivnosti za prejemanje paketov registrirajo sprejemnik `BroadcastReciever` (Izsek 3.3). Z registracijo se sprejemniku določi filter, na osnovi katerega bo sprejemal prihajajoče pakete. Sprejemnik implementira metodo `onRecive()`, ki iz prejetega paketa prebere sporočilo s podatki.

```
1 IntentFilter filter = new IntentFilter("current_speed");
2 BroadcastReceiver speedReceiver = new BroadcastReceiver() {
3
4   @Override
5   public void onReceive(Context context, Intent intent) {
6     speed = Math.ceil((double)intent.getExtras().get("speed"));
7   }
8
9 };
10 registerReceiver(speedReceiver, filter);
```

Izsek 3.3: Inicializacija in registracija sprejemnika `BroadcastReceiver`.

V aplikaciji so implementirane tri storitve. Prva skrbi za pridobivanje podatkov o lokaciji naprave, druga za podatke barometra in tretja za število opravljenih korakov.

Storitev za pridobivanje podatkov o lokaciji

Storitev skrbi za pridobivanje podatkov z uporabo razreda `LocationManager`, ki dostopa do sistemskih storitev za določanje lokacije naprave. Te storitve omogočajo periodično posodabljanje geografske lokacije. Za uporabo storitve je v datoteki `AndroidManifest.xml` treba dodati naslednjo pravico: `android.permission.ACCESS_FINE_LOCATION`.
Po preverjanju pravic `LocationManager` pošlje zahtevo za periodično pridobivanje lokacije s klicem metode `requestLocationUpdates()` . V zahtevi so definirani ponudnik storitev, časovni interval in minimalna razdalja med dvema lokacijama.
Na spremembe lokacije se odziva `LocationListener`, ki implementira metodo `onLocationChanged()` , ki kot argument prejme objekt tipa `Location`. Objekt vsebuje geografske koordinate, trenutno hitrost, smer potovanja, horizontalno natančnost v metrih in druge podatke. Zaradi velikega števila lokacij smo implementirali pogoj, ki določi, ali bo storitev posredovala prejeto lokacijo. Pogoj primerja vrednost horizontalne natančnosti nove lokacije in njeno oddaljenost od zadnje posredovane lokacije. Če je vrednost horizontalne natančnosti manjša od oddaljenosti, bo storitev posredovala novo lokacijo, v nasprotnem primeru pa bo počakala na naslednjo in ponovno izvedla primerjavo.
Storitev skrbi tudi za spremljanje stanja satelitov. Na spremembe stanja se odziva `GpsStatusListener`, ki implementira metodo `onGpsStatusChanged()`. Metoda prešteje število satelitov, ki se uporabljajo za določanje lokacije, in posreduje vrednost.

Barometer in števec korakov

Gre za dve ločeni storitvi, ki imata podobno implementacijo. Dostop do senzorjev naprave omogoča razred `SensorManager`, ki na osnovi vrste senzorja registrira ustrezni objekt `Listener`. Za uporabo barometra uporabi `Sensor.TYPE_PRESSURE`, za števec korakov pa `Sensor.TYPE_STEP_COUNTER`. V obeh primerih `Listener` implementira metodo `onSensorChanged()` , ki

kot argument prejme objekt tipa `SensorEvent`, v katerem je shranjena tabela vrednosti podatkovnega tipa `Float`. V primeru barometra iz tabele pridobimo vrednost zračnega pritiska v milibarih, števec korakov pa posreduje število opravljenih korakov od zadnjega izklopa naprave. V ta namen je ob zagonu storitve treba pridobiti trenutno število korakov in od vsake posredovane vrednosti odšteti prvotno. Vrednosti zračnega pritiska so do določene mere odvisne od proizvajalca senzorja.

3.4.3 Izračun naklona

Naklon je dodatni podatek, ki ga izračunamo po zaključku poti. Aplikacija izračuna povprečni naklon opravljene poti ali določenega dela. Za izračun naklona smo uporabili enačbo 3.1

$$\theta = \tan\left(\tan^{-1}\left(\frac{\Delta rise}{\Delta distance}\right)\right) \times 100, \tag{3.1}$$

kjer je θ vrednost naklona v odstotkih, $\Delta rise$ sprememba višine med dvema točkama, $\Delta distance$ pa sprememba razdalje [7].

3.4.4 Izračun porabljenih kalorij

Gre za približen izračun porabljenih kalorij pri teku in hoji, ker bi za uporabo natančnejše enačbe potrebovali podatke zunanjega merileca srčnega utripa. Izračun temelji na telesni teži uporabnika m, ki je nastavljena v nastavitvah aplikacije, času izvajanja aktivnosti t in vrednosti MET (angl. Metabolic equivalent) [12], ki je določena na osnovi povprečnega naklona poti in povprečne hitrosti. Za izračun porabljenih kalorij smo uporabili enačbo 3.2.

$$kcal = MET \times m \times t \tag{3.2}$$

3.5 Implementacija algoritmov

3.5.1 Algoritem za izračun razdalje med dvema točkama

Razdaljo med dvema točkama lahko izračunamo na različne načine. Izbira algoritma je odvisna od področja uporabe [24] in koordinatnega sistema. Najpogosteje uporabljena metoda za merjenje je evklidska razdalja na ravni črti med dvema točkama kartezijske ravnine. Razdaljo d izračunanamo z enačbo 3.3

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \tag{3.3}$$

kjer sta x_1 in x_2 horizontalni (X) koordinati obeh točk, y_1 in y_2 pa vertikalni (Y) koordinati.

Kadar koordinate točk merimo na sferi, zgornji izračun razdalje ni točen. V takšnem primeru lahko razdaljo med dvema točkama izmerimo na delu kroga, ki poteka skozi obe točki. Natančneje, kadar se vsaka točka meri z zemljepisno širino ali kotom do ekvatorialne ravnine (φ) in zemljepisno dolžino ali kotom med poldnevnikom točke in primarnim poldnevnikom (λ), lahko razdaljo med lokom izračunamo z enačbo 3.4

$$\cos\alpha = \sin\varphi_1 \sin\varphi_2 + \cos\varphi_1 \cos\varphi_2 \cos d\lambda, \tag{3.4}$$

kjer velja, da je $\mathrm{d}\lambda = |\ \lambda_1 - \lambda_2|$ absolutna razlika med zemljepisnima dolžinama obeh točk. V praksi se za izračun uporablja `Haversinova enačba` 3.5 in 3.6, ker se pri izračunu izognemo težavam pri ravnanju z negativnimi vrednostmi.

$$a = \sin^2\frac{d\varphi}{2} + \cos\varphi_1 \cos\varphi_2 \sin^2\frac{d\lambda}{2}, \tag{3.5}$$

$$c = 2 \times \arcsin(min(1, \sqrt{a})), \tag{3.6}$$

kjer funkcija `min` poskrbi, da vrednost `a` ni večja od 1 (zgornja meja funkcije `arcsin`). Vrednost c je centralni kot med točkama. Razdaljo nato izračunamo z enačbo 3.7

$$d = c \times R, \tag{3.7}$$

kjer vrednost R predstavlja polmer Zemlje v kilometrih. Končni rezultat na koncu pretvorimo v metre. Pomembno je omeniti, da je za uporabo zgornje enačbe treba pretvoriti zemljepisno širino in dolžino v radiane.
Algoritem smo implementirali v Javi (Izsek 3.4).

```
public double calculateDistance(LatLng location1, LatLng
    location2) {
  int R = 6371;

  double dLat = toRadians(location2.latitude - location1.
    latitude);
  double dLon = toRadians(location2.longitude - location1.
    longitude);
  double a = Math.pow(Math.sin(dLat / 2), 2) +
      Math.pow(Math.sin(dLon / 2), 2) *
      Math.cos(toRadians(location1.latitude)) *
      Math.cos(toRadians(location2.latitude));

  double c = 2 * Math.asin(Math.min(1, Math.sqrt(a)));
  return R * c * 1000;
}
```

Izsek 3.4: Implementacija Haversinovega algoritma v programskem jeziku Java.

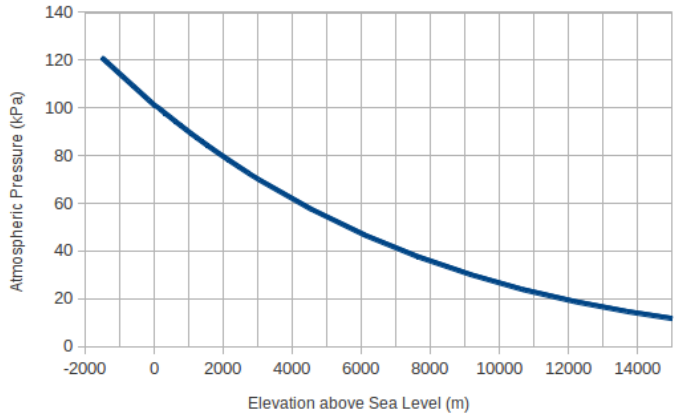
3.5.2 Algoritem za izračun nadmorske višine

Nadmorsko višino posameznih lokacij smo izračunali na osnovi stopnje zamika (angl. Laps Rate) [16] temperature pri vertikalnem gibanju skozi zemeljsko ozračje. Povprečna sprememba temperature znaša približno 6,5 °C na kilometer višine v nižjem pasu ozračja (troposfera). Korelacijo med nadmorsko višino, temperaturo in zračnim pritiskom lahko vidimo na sliki 3.2. Iz diagrama je razvidno, da se zračni pritisk pri večanju nadmorske višine

zmanjšuje in obratno. Na osnovi zgoraj omenjenih zvez lahko nadmorsko višino lokacije izračunamo z uporabo hipsometrične enačbe 3.8 [6]

$$h = \frac{((\frac{P_0}{P})^{\frac{1}{5.257}} - 1)(T + 273.15)}{0.0065}, \tag{3.8}$$

kjer je vrednost h nadmorska višina v metrih, P_0 vrednost zračnega pritiska na gladini morja, P vrednost zračnega pritiska na lokaciji in T temperatura na lokaciji. Vrednosti zračnega pritiska so podane v milibarih, temperatura pa v stopinjah Celzija. Enačba je učinkovita do nadmorske višine okrog enajst tisoč metrov, kjer spremembe zračnega pritiska na osnovi nadmorske višine in temperature niso več linearne.



Slika 3.2: Korelacija med nadmorsko višino, temperaturo in zračnim pritiskom.

Algoritem smo implementirali v Javi (Izsek 3.5).

```
1 public void calculateElevation() {
2   double elevation = (
3     (((Math.pow(((this.weatherData.getPressureAtSeaLevel()) /
    this.airPressure), (1 / 5.257))) - 1) *
4     (this.weatherData.getTemperature() + 273.15)) / 0.0065
5   );
6
```

```
7   this.elevation = round(elevation, 2);
8 }
```

Izsek 3.5: Implementacija enačbe za izračun nadmorske višine.

3.5.3 Algoritem za kodiranje zapisa koordinat

Algoritem za kodiranje koordinat omogoča zapis serije zaporednih lokacij v en niz in s tem bistveno zmanjša število potrebnih znakov za zapis poti v datoteko. Deluje na osnovi kompresije z izgubo (angl. Lossy Compression), ki iz geografskega zapisa lokacije odstrani odvečne vrednosti decimalnega dela zemljepisne širine in dolžine. Decimalni del teh vrednosti je običajno določen z osmimi do petnajstimi števkami, za pravilen izris lokacije na zemljevidu pa je potrebnih le prvih pet. Algoritem dodatno varčuje s prostorom, tako da vsaki lokaciji razen prvi zakodira le odmik od prejšnje lokacije. Za izvedbo kodiranja najprej pretvori decimalni zapis koordinate v binarno vrednost, iz katere nato pridobi serijo znakov ASCII na osnovi kodirne sheme `base64`. Da bi zagotovil pravilno prikazovanje znakov, se vsaki kodirani vrednosti pred pretvorbo v ASCII prišteje vrednost 63 (koda za znak ?).

Potek kodiranja

Algoritem najprej vzame vrednost zemljepisne širine (za primer bomo vzeli vrednost –179.9832104), jo pomnoži s faktorjem $1e^5$ in zaokroži rezultat (3.9).

$$-179.9832104 \times 1e^5 = -17998321 \tag{3.9}$$

Desetiško vrednost nato pretvori v binari zapis. Če je število negativno, mora biti pretvorba izvedena z uporabo dvojiškega komplementa (3.10).

$$-17998321_{(DEC)} = 11111110111011010101111000001111_{(BIN)} \tag{3.10}$$

Po pretvorbi, binarno število (3.10) zamakne v levo za en bit:
$11111101110110101011110000011110_{(BIN)}$.

Če je izhodiščna decimalna vrednost negativna (kot v tem primeru), bo algoritem izvedel inverz binarnega števila. Inverz spremeni vse bite z vrednostjo `0` $\longrightarrow$ `1` in `1` $\longrightarrow$ `0`:
$000000100010010101000011111000 01_{(BIN)}$.

Binarno število nato od desne proti levi razdeli na 5-bitne dele in jih postavi v obratni vrstni red:
00001 00010 01010 10000 11111 00001,
00001 11111 10000 01010 00010 00001.

Nato nad posameznimi deli izvede operacijo `OR`, zadnji 5-bitni del pa izpusti: 100001 111111 110000 101010 100010 000001. Na koncu vsak del pretvori v desetiško vrednost in prišteje 63. Pridobljene desetiške vrednosti nato pretvori v znake ASCII, ki skupaj tvorijo niz **‘˜oia@**. Postopek se nato ponovi še za zemljepisno dolžino.
Primer za lokacijo FRI: (46.050211, 14.469026) $\rightarrow$ **ydaxGm˜hwA**.
Algoritem smo implementirali v Javi (Izsek 3.6) na osnovi primera iz Googlove dokumentacije [5].

```
1  public String encodeCoordinates() {
2    long lastLat = 0;
3    long lastLng = 0;
4    String encodedString = "";
5    for (Coordinate coordinate : this.coordinates) {
6      long lat = Math.round(coordinate.getLatitude() * 1e5);
7      long lng = Math.round(coordinate.getLongitude() * 1e5);
8      long dLat = lat - lastLat;
9      long dLng = lng - lastLng;
10     encodedString += encode(dLat);
11     encodedString += encode(dLng);
12     lastLat = lat;
13     lastLng = lng;
14   }
15   return encodedString;
16 }
```

```
17 private String encode(long value) {
18   StringBuffer result = new StringBuffer();
19   if (value < 0) {
20     value = ~(value << 1);
21   } else {
22     value = value << 1;
23   }
24   while (value >= 0x20) {
25     result.append(Character.toChars((int) ((0x20 | (value & 0x1f
     )) + 63)));
26     value = value >> 5;
27   }
28   result.append(Character.toChars((int) (value + 63)))
29   return result.toString();
30 }
```

Izsek 3.6: Implementacija algoritma za kodiranje koordinat v programskem jeziku Java.

3.6 Shranjevanje podatkov

Po končanem spremljanju poti aplikacija obdelane podatke zapiše v datoteko. Za shranjevanje smo izbrali format JSON, ker je pregleden in enostaven za uporabo. Celotna vsebina datoteke je zapisana v objekt tipa `JSONObject`, ki vsebuje štiri objekte tipa `JSONArray`. Prvi hrani splošne podatke, kot so dolžina poti, porabljeni čas, podatki o nadmorski višini in hitrosti. Drugi vsebuje vremenske podatke. Tretji vsebuje nize kodiranih geografskih koordinat posameznih odsekov poti. Specifične podatke posamezne lokacije hrani četrti objekt. Za vsako lokacijo se shranjujejo nadmorska višina, hitrost, časovna značka in vmesna razdalja.

3.7 Aktivnosti

V tem poglavju bodo predstavljene aktivnosti aplikacije. Pogledali si bomo funkcionalnosti, ki so implementirane v okviru posamezne aktivnosti, njeno delovanje in izgled uporabniškega vmesnika med izvajanjem.

3.7.1 MainMenuActivity

Namen aktivnosti:

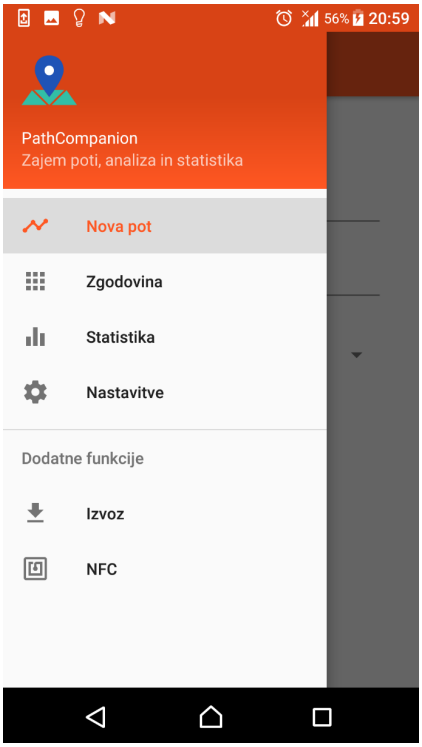
- izhodišče za navigacijo znotraj aplikacije;
- priprava novega zajema poti;
- prikaz zgodovine in statistike shranjenih poti;
- izvoz shranjene poti.

Struktura aktivnosti:

- `MainMenuActivity.java` – skrbi za delovanje aktivnosti in prikaz uporabniškega vmesnika.
- Podporni razredi, ki skrbijo za delovanje funkcionalnosti v fragmentih in prikaz elementov njihovega uporabniškega vmesnika.

Delovanje

Osnovna komponenta aktivnosti je glavni meni. Za implementacijo smo uporabili gradnik `Navigation Drawer`, ki prikazuje glavne navigacijske možnosti aplikacije na levem robu zaslona (Slika 3.3). Plošča z opcijami menija je dostopna s potegom (angl. Swipe) od leve proti desni strani zaslona ali s klikom ikone na levi strani orodne vrstice. Meni je večino časa skrit, kar maksimira prostor za prikaz vsebine. Aktivnost vsebuje štiri funkcionalnosti, ki so implementirane s fragmenti. Ob izbiri opcije v meniju se vsebinski del uporabniškega vmesnika aktivnosti nadomesti z uporabniškim vmesnikom fragmenta posamezne funkcionalnosti.



Slika 3.3: Glavni meni aplikacije.

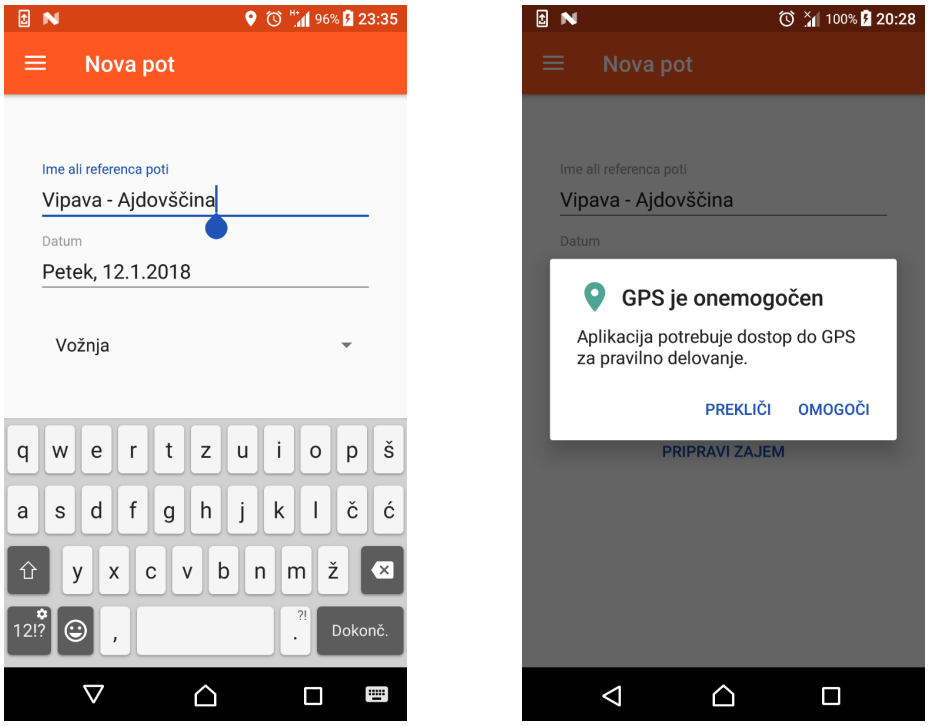
NewPathFragment

Vsebina fragmenta postane aktivna, ko uporabnik v glavnem meniju (Slika 3.3) izbere opcijo `Nova pot`. Namen modula:

- pridobitev parametrov za definicijo poti;
- preverjanje dostopa do lokacije naprave in interneta;
- prenos parametrov v aktivnost CaptureActivity.

Modul pridobi referenco, datum in vrsto aktivnosti (Slika 3.4a), ki določi način spremljanja poti. Referenca, datum in vrsta določajo del imena datoteke, v kateri bodo po končanem spremljanju shranjeni podatki. Datum in vrsto uporabljajo tudi filtri, s katerimi lahko uporabnik omeji prikaz zgodovine gibanja. Pred prenosom parametrov v aktivnost CaptureActivity se

izvede preverjanje dostopa do lokacije mobilne naprave in stanja internetne povezave. Če so vsi pogoji za pripravo zajema izpolnjeni, se ob kliku na gumb požene aktivnost CaptureActivity, v naspotnem primeru pa se prikaže ustrezno obvestilo, ki nakazuje, kateri od pogojev ni bil izpolnjen (Slika 3.4b).



(a) Vnos osnovnih parametrov. (b) Onemogočena uporaba lokacije.

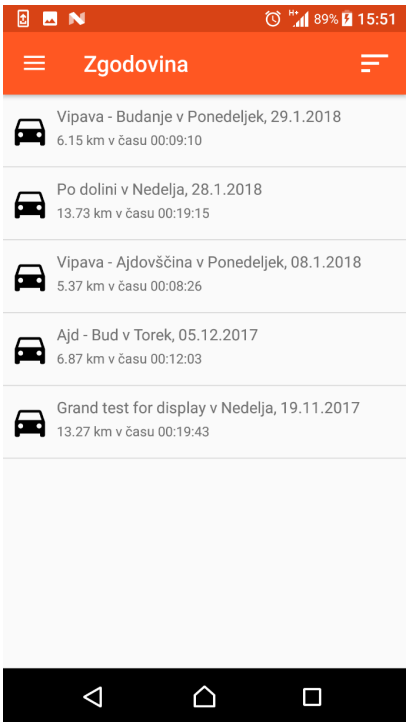
Slika 3.4: Vnos osnovnih podatkov in prikaz obvestila v primeru napake.

HistoryFragment

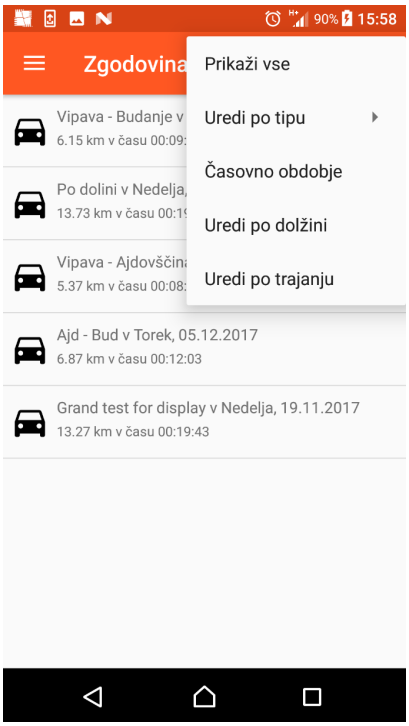
Vsebina fragmenta postane aktivna, ko uporabnik v glavnem meniju (Slika 3.3) izbere opcijo `Zgodovina`. Namen modula:

- prikaz zgodovine gibanja;
- prehod na izvedbo analize poti;
- izbris poti iz zgodovine.

Modul prikaže zgodovino gibanja osebe v obliki urejenega seznama, v katerem so elementi urejeni po datumu. Za implementacijo seznama smo uporabili gradnik `ListView` (Slika 3.5a). Vsebino seznama na osnovi definicije izgleda v datoteki `list_adapter.xml` pripravi prilagojen `ArrayAdapter`. Privzeti prikaz vsebuje celotno zgodovino gibanja, uporabnik pa lahko prikaz prilagodi z uporabo filtrov, ki se nahajajo v meniju na desni strani orodne vrstice (Slika 3.5b).



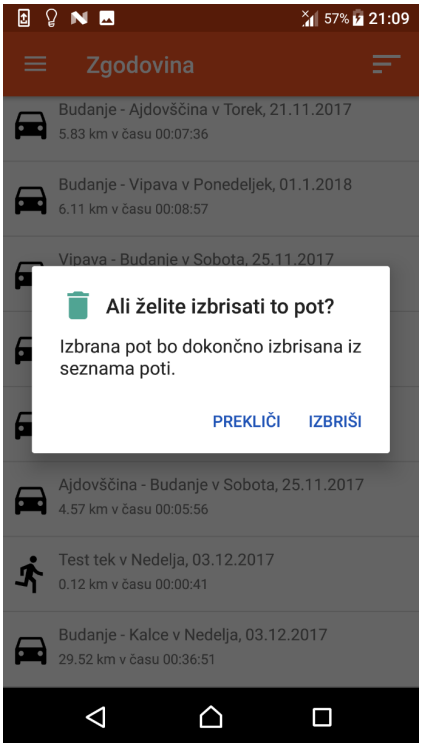
(a) Seznam zgodovine gibanja.



(b) Meni s filtri za prikaz vsebine.

Slika 3.5: Prikaz zgodovine in filtri za prikaz vsebine.

Vsi elementi seznama so interaktivni in imajo definirani dve akciji. S klikom na element se požene aktivnost, ki izvede analizo poti in prikaže rezultate, dolgi klik pa sproži akcijo za izbris poti iz zgodovine (Slika 3.6).



Slika 3.6: Prikaz opozorila pred izbrisom poti.

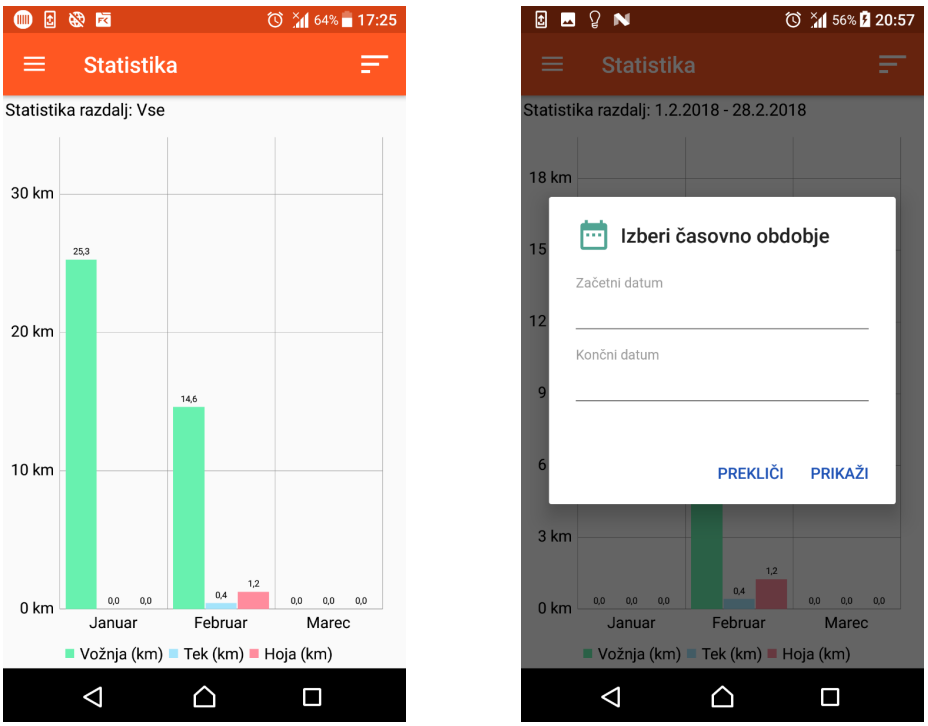
StatisticsFragment

Vsebina fragmenta postane aktivna, ko uporabnik v glavnem meniju (Slika 3.3) izbere opcijo `Statistika`. Namen modula:

- pregled statistike shranjenih poti;
- možnost primerjave statistike glede na časovno obdobje;
- spremljanje napredka in zadanih ciljev.

StatisticsFragment omogoča statistični pregled določenih parametrov celotne zgodovine shranjenih poti ali točno določenega časovnega obdobja. Za prikaz podatkov je bila uporabljena zunanja knjižnica `MPAndroidChart` [11] za vizualizacijo podatkov, ki podpira osem različnih vrst grafov. Za prikaz grafa je v datoteki XML, ki definira izgled uporabniškega vmesnika, treba dodati

posebni gradnik. Graf lahko z uporabo dodatnih objektov prilagodimo za potrebe prikaza različnih podatkov. Za prikaz statističnih podatkov poti je bil uporabljen stolpni graf (Slika 3.7a), ki smo ga implementirali z uporabo gradnika `BarChart` in dveh objektov za spreminjanje osi: `XAxis` in `YAxis`.



(a) Statistika razdalj.

(b) Statistika razdalj za februar 2018.

Slika 3.7: Prikaz statistike razdalj vseh shranjenih poti in meni za izbiro časovnega obdobja.

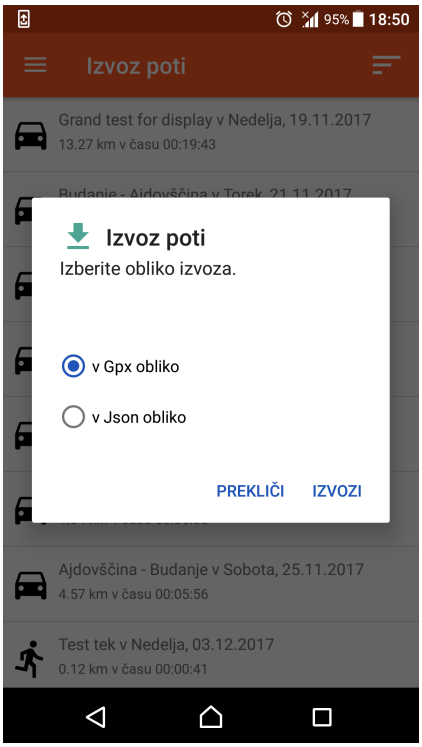
Na grafu so podatki prikazani za vsak mesec posebej, vendar so zaradi velikosti zaslona na enkrat vidni le podatki treh mesecev, za ogled ostalih pa je potreben sprehod po grafu (angl. Swipe) v desno ali levo smer. Uporabnik lahko v meniju izbira med štirimi statističnimi podatki: razdalja, število korakov, število porabljenih kalorij in odstotek aktivnosti. Poleg izbire načina prikaza je v meniju tudi opcija za izbiro časovnega obdobja (Slika 3.7b).

ExportFragment

Vsebina fragmenta postane aktivna, ko uporabnik v glavnem meniju (Slika 3.3) izbere opcijo `Izvoz`. Namen modula:

- izvoz shranjene poti v datoteko JSON ali GPX (angl. Global Positioning System Exchange Format).

ExportFragment omogoča izvoz shranjene poti v dveh formatih: JSON in GPX (Slika 3.8). Izvoz v formatu JSON je namenjen predvsem arhiviranju oziroma ustvarjaju varnostne kopije v primeru izgube podatkov mobilne naprave. Struktura datoteke JSON, ki nastane pri izvozu, je enaka kot pri shranjevanju nove zajete poti.



Slika 3.8: Izbira formata za izvoz shranjene poti.

Izvoz poti v format GPX je drugačen, saj vsebuje le geografske koordinate, nadmorsko višino in časovno značko posameznih lokaciji. Omogoča pregled

poti s programskimi orodji, kot je na primer Google Earth Pro, in možnost uporabe v drugih aplikacijah ter napravah.
Za zapis poti v format GPX je bila uporabljena zunanja knjižnica `GPXParser` [10]. Knjižnica omogoča uporabo razreda `GPX.java`, ki ustvari objekt, v katerem se shranjujejo podatki poti pred zapisom v datoteko. Sama pot je definirana z objektom tipa `Track`. Vsebuje ime poti, ki ga doda metoda `setName()` , vrsto poti, ki jo doda metoda `setType()` glede na to ali gre za vožnjo, hojo ali tek, in seznam lokacij, ki ga doda metoda `setTrackPoints()`. Vsaka lokacija v seznamu je definirana z objektom tipa `Waypoint` in vsebuje geografske koordinate, nadmorsko višino in časovno značko zajema lokacije v naslednji obliki:

```
2018-01-28T16:37:45Z
```

Celotna pot se nato s klicem metode `addTrack()` zapiše v objekt tipa `GPX`. Za zapis objekta poti v datoteko poskrbi objekt `GPXParser` s klicem metode `writeGPX()`, ki prejme dva argumenta: objekt poti in izhodni tok `FileOutputStream`. Za lažjo berljivost datoteke poskrbi razred `Transformer`, ki vsebino strukturira, kot je prikazano v izseku 3.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx
  xmlns="http://www.topografix.com/GPX/1/1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.1"
  creator="RouteTracker"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1
  http://www.topografix.com/GPX/1/1/gpx.xsd">
   <trk>
    <name>Budanje - Vipava</name>
    <type>Driving</type>
    <trkseg>
      <trkpt lat="45.87478" lon="13.94815">
        <ele>242.0</ele>
        <time>2018-01-28T16:37:45Z</time>
```

```
16        </trkpt>
17
18        <!-- vmesne lokacije -->
19
20        <trkpt lat="45.8743" lon="13.94908">
21          <ele>240.0</ele>
22          <time>2018-01-28T16:56:45Z</time>
23        </trkpt>
24      </trkseg>
25    </trk>
26 </gpx>
```

Izsek 3.7: Urejen zapis v datoteki GPX.

3.7.2 CaptureActivity

Namen aktivnosti:

- izračun in beleženje parametrov poti;
- prikaz določenih parametrov poti v realnem času;
- zapis parametrov v datoteko po koncu beleženja.

Struktura aktivnosti:

- `CaptureActivity.java` – skrbi za delovanje aktivnosti in prikaz uporabniškega vmesnika.
- `PathData.java` – razred za izračun in shranjevanje parametrov poti med spremljanjem.
- `Coordinate.java` – na osnovi razreda aktivnost ustvari objekt tipa `Coordinate`, v katerem je definirana posamezna lokacija.

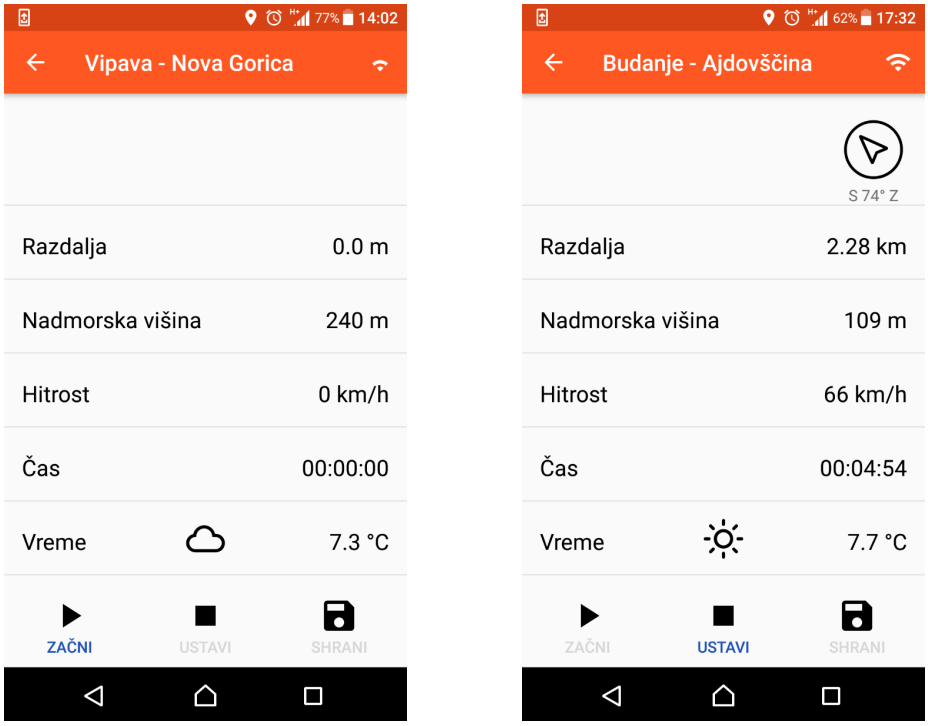
Delovanje

Po prehodu v aktivnost CaptureActivity se v metodi `onCreate()` inicializirajo spremenljivke in objekti. Sledi zagon storitev za pridobivanje podatkov iz senzorjev in sistema GPS ter registriracija sprejemnikov za pridobitev njihovih vrednosti. Pred začetkom spremljanja poti mora storitev za lokacijo najti vsaj štiri satelite. Nato vrne začetno lokacijo, ki se uporabi v zahtevi za pridobitev vremenskih podatkov ter izračun koeficienta za kalibracijo vrednosti barometra. Koeficient k se izračuna z enačbo 3.11 [8]

$$k = P - P_0 \left(1 - \frac{0.0065h}{T + 273.15 + 0.0065h}\right)^{5.257}, \tag{3.11}$$

kjer je P vrednost zračnega pritiska barometra, P_0 vrednost zračnega pritiska na gladini morja, T temperatura in h nadmorska višina. Nadmorsko višino začetne lokacije pošlje vmesnik `GoogleElevation` na osnovi geografskih koordinat. Koeficient k se nato prišteje k vsaki vrednosti barometra. Po inicializaciji se na uporabniškem vmesniku prikažejo podatki izhodiščne lokacije (Slika 3.9a).

Uporabnik lahko na tem mestu začne spremljati pot s klikom na gumb `ZAČNI`. Aktivnost ob prejemu nove lokacije pokliče metodo `addCoordinate()` podpornega razreda `PathData.java`, ki kot parameter prejme lokacijo, zračni pritisk barometra, časovno značko, hitrost in indeks vremenskih podatkov. Metoda na osnovi prejetih parametrov ustvari objekt tipa `Coordinate`, ki hrani podatke o lokacij. Znotraj objekta se izvedeta metoda `calculateDistance()`, ki na osnovi `Haversinovega` algoritma (poglavje 3.5.1) izračuna razdaljo od predhodne koordinate, in metoda `calculateElevation()`, ki na osnovi algoritma iz poglavja 3.5.2 izračuna nadmorsko višino.

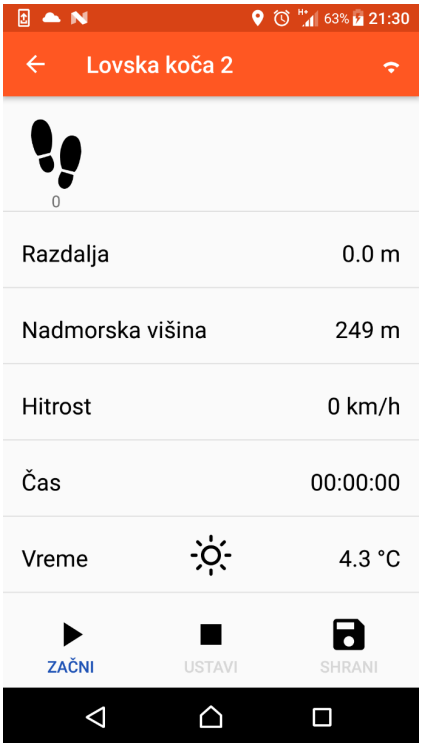


(a) Po inicializaciji. (b) Med spremljanjem poti.

Slika 3.9: Izgled uporabniškega vmesnika aktvnosti za spremljanje poti.

Objekt `PathData` glede na vrsto poti pošlje zahtevo za posodobitev vremenskih podatkov. Med spremljanjem vožnje se to zgodi vsakih pet kilometrov, med tekom in hojo pa vsak kilometer. Aktivnost periodično izvaja pregled shranjenih koordinat. Periodično se ustvari objekt razreda `ElevationCorrection.java`, ki ga razširja razred `AsyncTask`. Objekt prejme zadnjih 80 lokacij in sestavi zahtevo za vmesnik `GoogleElevation`, ki vrne odgovor z nadmorskimi višinami lokacij v zahtevi. Vrednosti nadmorskih višin se nato primerjajo, in kjer je treba, ustrezno popravijo. Pri spremljanju vožnje se izvede tudi korekcija geografskih koordinat. Lokacije, ki jih posreduje GPS, običajno niso postavljene točno na poti, po kateri potujemo. Za reševanje problema smo uporabili vmesnik `GoogleRoads` v povezavi s funkcijo `Snap to Roads`, ki prejete koordinate postavi na cesto. Zahtevo za vmesnik sestavi objekt razreda `SnapToRoads.java` z razširitvijo `AsyncTask`

na osnovi zadnjih 80 lokacij. Vmesnik na podano zahtevo odgovori s popravljenimi koordinatami, ki se nato ustrezno nastavijo. Vmesni pregled se zaključi s kodiranjem zapisa koordinat. Kodiranje izvede objekt razreda `PolylineEncoder.java` z algoritmom, ki smo ga spoznali v poglavju 3.5.3. Med spremljanjem poti se na uporabniškem vmesniku v realnem času izpisujejo dolžina opravljene poti, trenutna nadmorska višina, čas trajanja od začetka poti, temperatura, smer potovanja (Slika 3.9b), v primeru spremljanja teka ali hoje pa tudi število opravljenih korakov (Slika 3.10). Spremljanje poti traja, dokler ga uporabnik ne prekine s klikom na gumb `USTAVI`. Ko je spremljanje končano, aktivnost ustavi storitve za pridobivanje podatkov in sprejemnike. Na tem mestu lahko uporabnik shrani zapis poti v datoteko, ali pa začne s spremljanjem nove poti.



Slika 3.10: Uporabniški vmesnik pred začetkom spremljanjem teka ali hoje.

3.7.3 DisplayDataActivity

Namen aktivnosti:

- prikaz statičnih parametrov zajete poti;
- izris poti na zemljevidu;
- prikaz določenih parametrov na grafu.

Struktura aktivnosti:

- `DisplayDataActivity.java` – skrbi za delovanje aktivnosti in prikaz uporabniškega vmesnika.
- `Data.java` - razred, ki implementira asinhronsko operacijo za pridobitev podatkov shranjene poti iz datoteke.
- `PolylineDecoder.java` - razred, ki dekodira niz kodiranih geografskih koordinat.

Delovanje

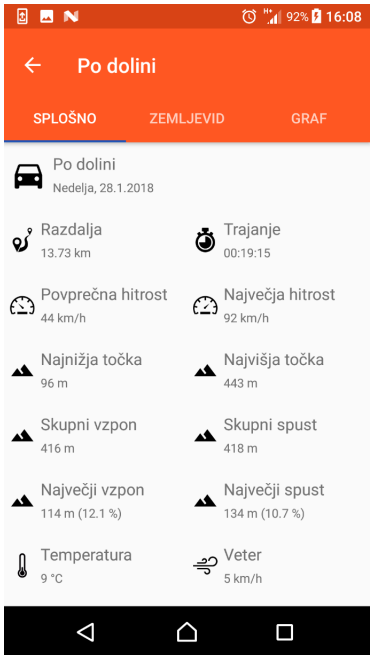
Aktivnost DisplayDataActivity vizualizira vse parametre poti, ki jih med zajemom beleži aktivnost CaptureActivity, in izvede določene analize. Zaradi boljše preglednosti podatkov smo prikaz razdelili na tri module. Vsak modul je implementiran znotraj fragmenta, ki skrbi za pravilen prikaz vsebine na zaslonu in uporabo pripadajočih funkcij. Vsi moduli se nahajajo v gradniku `TabLayout`. Takšen način postavitve razdeli prikazane podatke v zavihke in omogoča enostavno prehajanje med njimi.
Vsakemu modulu so ob nastanku dodeljeni ustrezni podatki, ki jih iz datoteke shranjene poti prebere podporni razred `Data.java`. Datoteka, iz katere bo razred pridobil podatke, je določena ob izbiri poti v seznamu shranjenih poti fragmenta `HistoryFragment`.

OverviewFragment

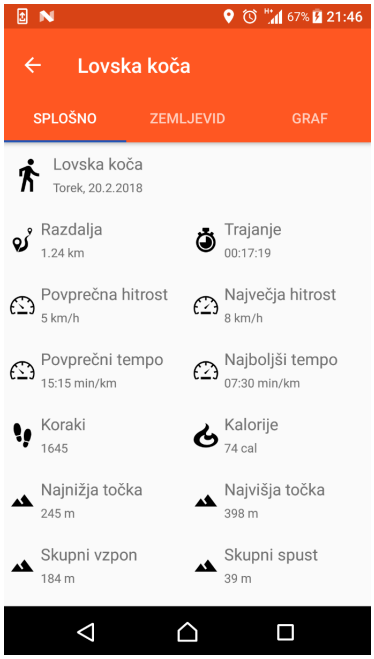
Namen modula:

- prikaz parametrov zajete poti.

OverviewFragment prikazuje statične podatke poti. Prikaz vsebuje razdaljo, čas trajanja, povprečno in največjo hitrost, informacije o nadmorski višini, spustih, vzponih, temperaturi in hitrosti vetra. Prikaz teka in hoje poleg naštetih parametrov vsebuje še povprečni in najboljši tempo, število opravljenih korakov in približno število porabljenih kalorij.
Za razporeditev prikaza podatkov je bila uporabljena tabelarična postavitev (angl. TableLayout) z dvema stolpcema in ustreznim številom vrstic glede na vrsto poti. Vsak prikazan podatek ima definiran svoj gradnik za prikaz ikone (angl. ImageView) in dve besedilni polji (angl. TextView) za naziv in vrednost (Slika 3.11).



(a) Prikaz parametrov vožnje.



(b) Prikaz parametrov hoje.

Slika 3.11: Prikaz statičnih parametrov vožnje in hoje.

MapFragment

Namen modula:

- izris poti na zemljevidu;
- grafični prikaz hitrosti, nadmorske višine, spustov in vzponov na odsekih izrisane poti.

MapFragment skrbi za prikaz zemljevida in izris poti. Za implementacijo zemljevida je bil uporabljen Google Maps API, ki omogoča uporabo objekta tipa `GoogleMap` in povratnega klica `OnMapReadyCallback`. Povratni klic implementira metodo `onMapReady()` , ki se izvede, ko je objekt zemljevida pripravljen za uporabo. Metoda `onMapReady()` najprej konfigurira zemljevid glede na nastavitve izgleda aplikacije in nato ustvari objekt tipa `PolylineOptions`, ki na osnovi seznama zajetih lokacij sestavi pot, ki bo izrisana na zemljevidu (Izsek 3.8).

```
polylineOptions = new PolylineOptions();
polylineOptions.width(13);
polylineOptions.color(Color.BLUE);
for (int i = 0; i < coordinates.size(); i++) {
  latLng = coordinates.get(i).split(Pattern.quote(","));
  polylineOptions.add(
      new LatLng(
          Double.parseDouble(latLng[0]),
          Double.parseDouble(latLng[1])
      )
  );
}
googleMapObject.addPolyline(polylineOptions);
googleMapObject.moveCamera(
    CameraUpdateFactory.newLatLngZoom(
        normal.getPoints().get(0), 14));
```

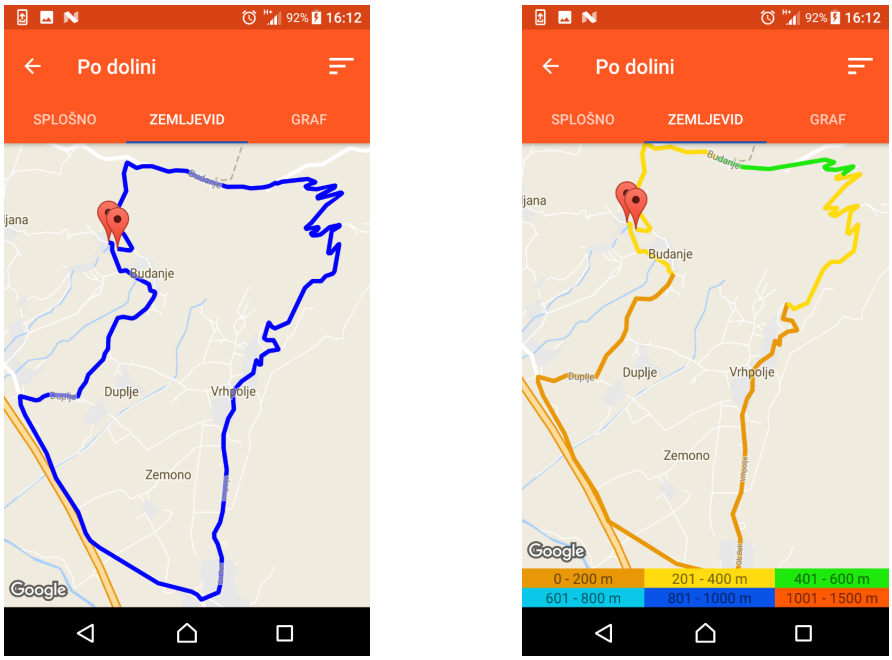
Izsek 3.8: Koda, ki na osnovi koordinat izriše pot na zemljevidu.

Seznam podatkov je podan kot argument `Bundle`, ko aktivnost DisplayDataActivity ustvari `MapFragment`. Vsebuje koordinatni zapis, nadmorsko višino

in hitrosti na vsaki zajeti lokaciji. `MapFragment` izriše pot na štiri načine, ki so definirani v meniju na desni strani orodne vrstice:

- **navadni prikaz** izriše samo opravljeno pot (Slika 3.12a);
- **prikaz hitrosti** izriše opravljeno pot v več odsekih, glede na hitrost gibanja (Slika 3.13a);
- **višinski prikaz** izriše opravljeno pot v več odsekih, glede na nadmorsko višino (Slika 3.12b);
- **prikaz vzpona/spusta** izriše opravljeno pot v več odsekih, glede na vzpon ali spust (Slika 3.13b).

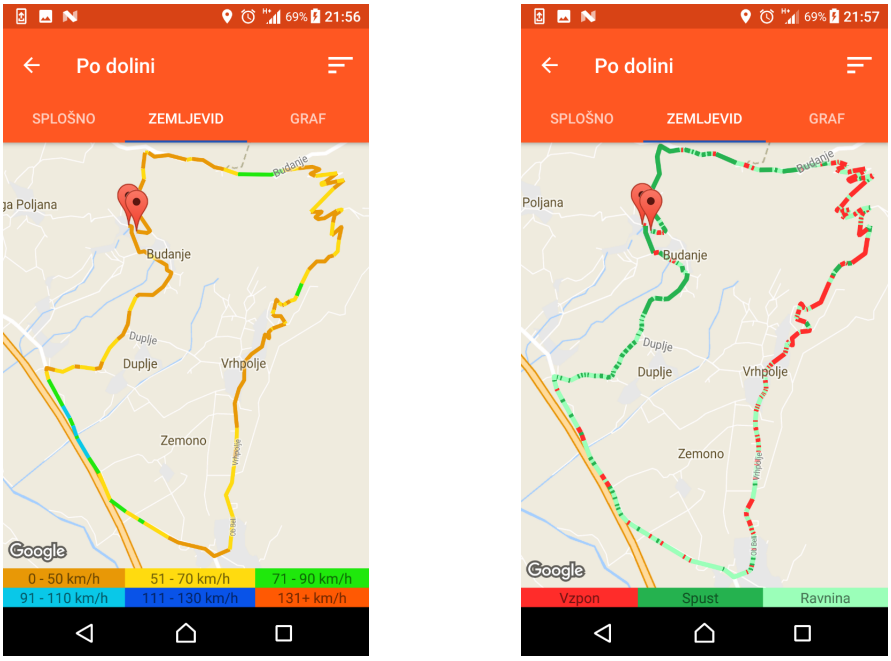
Ob prikazu hitrosti, nadmorske višine ter vzpona in spusta se pod zemljevidom prikaže legenda, ki opredeljuje prikaz posameznih odsekov poti. Začetek in konec poti je označen z objektom tipa `Marker`.



(a) Navadni izris poti.

(b) Izris glede na nadmorsko višino.

Slika 3.12: Navadni izris poti in izris z nadmorsko višino.



(a) Izris glede na hitrost.

(b) Izris glede na naklon.

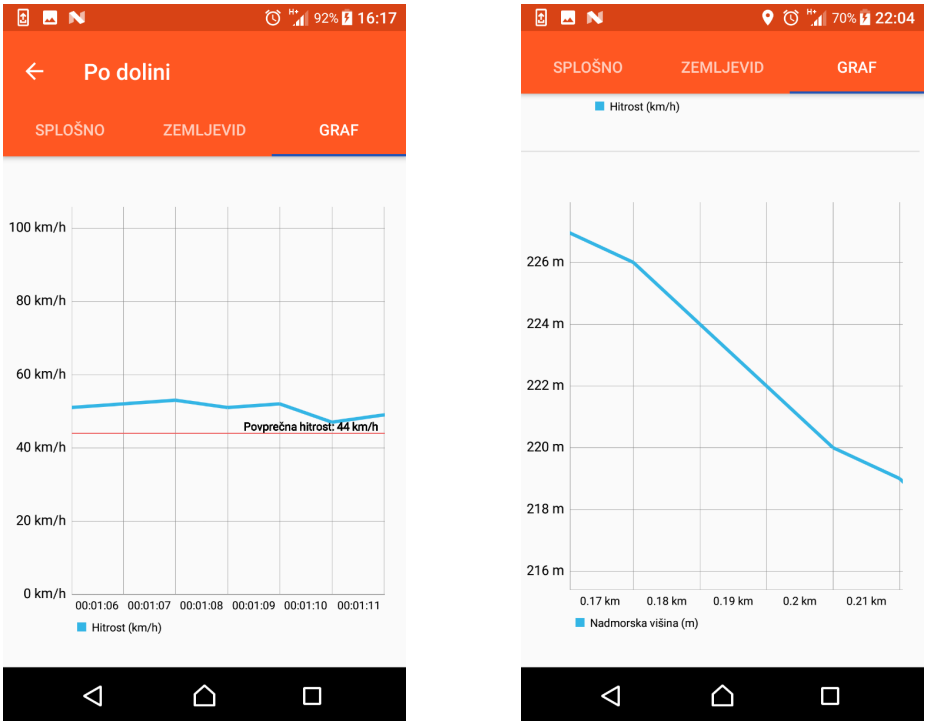
Slika 3.13: Izris poti glede na hitrost in naklon.

GraphFragment

Namen modula:

- prikaz spreminjanja hitrosti v odvisnosti od časa;
- prikaz spreminjanja nadmorske višine v odvisnosti od poti.

GraphFragment prikazuje spreminjanje hitrosti gibanja (Slika 3.14a) in nadmorske višine (Slika 3.14b) v odvisnosti od poti na grafu. Za implementacijo grafov smo uporabili zunanjo knjižnico `MPAndroidChart`, ki smo jo že spoznali pri prikazu statistike, vendar smo v tem primeru uporabili linearni graf. Za prikaz smo izbrali graf hitrosti v odvisnosti od časa in graf nadmorske višine v odvisnosti od razdalje. Na grafu hitrosti je z uporabo objekta tipa `LimitLine` prikazana tudi povprečna hitrost.



(a) Graf hitrosti v odvisnosti od časa. (b) Graf višine v odvisnosti od razdalje.

Slika 3.14: Prikaz hitrosti in nadmorske višine na grafu.

3.7.4 NfcActivity

Namen aktivnosti:

- odkrivanje pasivne značke NFC;
- priprava sporočila NDEF;
- zapis sporočila na značko NFC.

Struktura aktivnosti:

- `NfcActivity.java` – skrbi za delovanje aktivnosti in prikaz uporabniškega vmesnika.

Aktivnost omogoča zapis podatkov, ki so potrebni za zagon aplikacije na značko NFC v obliki sporočila NDEF. Po zapisu se aplikacija samodejno

zažene, ko uporabnik napravo približa znački (1 do 5 cm). Za zapis sporočila NDEF poskrbi objekt `NfcAdapter`. Pred pripravo sporočila aktivnost preveri, ali je na mobilni napravi omogočena storitev NFC in v nasprotnem primeru prikaže `Snackbar` obvestilo. Ko je storitev NFC omogočena, `NfcAdapter` registrira razpečevalec (angl. ForegroundDispatch), ki se odzove, ko naprava odkrije značko NFC. Ko razpečevalec odkrije značko, se izvede metoda, ki ustvari sporočilo NDEF (Izsek 3.9).

```
public NdefMessage createApplicationRecord(String packageName) {
  NdefMessage ndefMessage = new NdefMessage(
      new NdefRecord[]{
          NdefRecord.createApplicationRecord(packageName)
      }
  );

  return ndefMessage;
}
```

Izsek 3.9: Metoda, ki ustvari sporočilo NDEF.

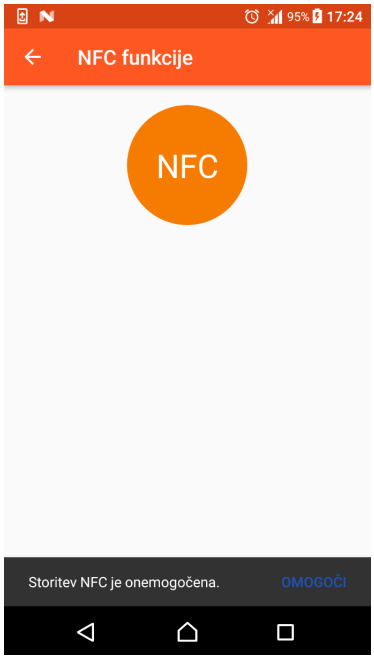
Sporočilo NDEF [18] je sestavljeno iz enega ali več zapisov (angl. NDEF Record). Zapis je enota za prenos podatkov v okviru sporočila. Prvi zapis je označen z zastavico MB (angl. Message Begin), ki označuje začetek sporočila. Sledijo mu zapisi, ki vsebujejo podatke, s katerimi je določeno, kako se bo naprava odzvala ob odkritju značke. V tem primeru se v sporočilo (Slika 3.15) zapiše ime paketa aplikacije.

NDEF Message		
R_1 MB=1	**R_n = com.tracker.path.pathtracker**	R_t ME=1

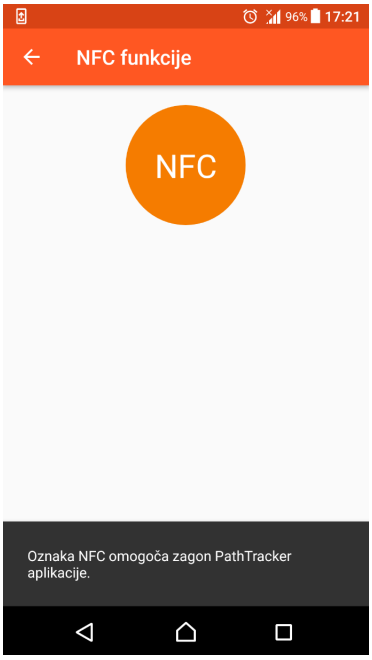
Slika 3.15: Oblika sporočila NDEF za zagon aplikacije.

Zapis vsebuje paket aplikacije tipa AAR (angl. Android Application Record). To je posebna vrsta zapisa, ki operacijskemu sistemu Android pove, da mora značko NFC s takšnim zapisom prevzeti točno določena aplikacija. Zadnji

zapis je označen z zastavico ME (angl. Message End), ki označuje konec sporočila.



(a) Onemogočena storitev NFC.



(b) Uspešen zapis sporočila NDEF.

Slika 3.16: Primer obvestila med zapisom sporočila NDEF.

Pred zapisom sporočila se izvede metoda `connect()` , ki omogoči izvedbo vhodnih/izhodnih operacij nad značko NFC. Aktivnost nato preveri, ali značka dovoljuje zapisovanje podatkov in velikost njenega pomnilnika ter preveri ali na znački že obstaja kakšen zapis. Če značka že vsebuje zapis, se izvede metoda `formatNdef()` , ki najprej formatira vsebino značke. Ko so vsi pogoji za zapis izpolnjeni, se izvede metoda `writeNdefMessage()` , ki na značko zapiše sporočilo NDEF (Slika 3.16a), v nasprotnem primeru pa se prikaže ustrezno obvestilo o napaki (Slika 3.16b).

3.7.5 SettingsActivity

Namen aktivnosti:

- nastavitve izgleda aplikacije;
- izbira jezika;
- vnos teže za izračun porabljenih kalorij.

Struktura aktivnosti:

- `SettingsActivity.java` – skrbi za delovanje aktivnosti in prikaz uporabniškega vmesnika.

Aktivnost SettingsActivity temelji na vmesniku za dostop in spreminjanje nastavitev (angl. SharedPreferences). Sestavljene so iz parov ključnih vrednosti[1] (angl. Key/Value pair) primitivnih podatkovnih tipov, ki so shranjeni v datoteki znotraj datotečnega sistema aplikacije. Nastavitve so prikazane z različnimi gradniki uporabniškega vmesnika.
Za uveljavljanje nastavitev skrbi `OnPreferenceChangeListener`, ki se sproži ob spremembi izbire. Do shranjenih nastavitev lahko kadarkoli dostopa vsaka aktivnost.

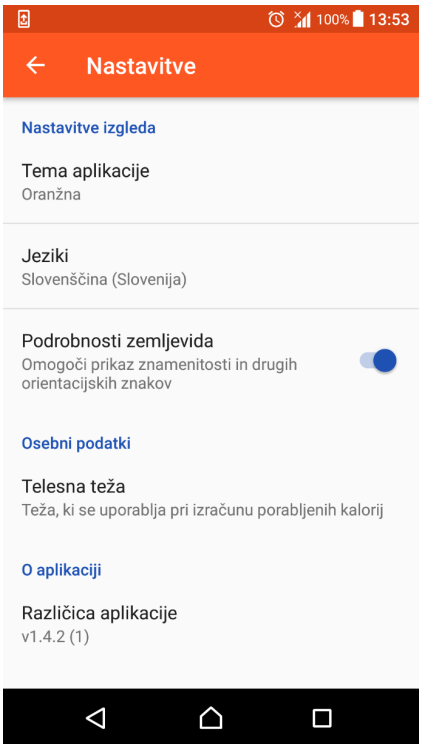
Posamezne nastavitve
Aplikacija ima štiri nastavitve, ki so razdeljene v dve kategoriji (Slika 3.17). Prva kategorija vsebuje nastavitve izgleda:

- Tema aplikacije določa barvno shemo gradnikov uporabniškega vmesnika. Nastavitev je zgrajena v obliki seznama, ki vsebuje šest različnih barvnih shem.
- Jezik nadzira prikaz besedila v posameznih aktivnostih, kot so naslovi, obvestila, opcije v meniju. Nastavitev jezika ne vpliva na jezik tipkovnice pri vnosu besedila in izgled koledarja za izbiro časovnega obdobja.

[1] Key/Value pair: Niz dveh povezanih podatkovnih elementov.

Tipkovnica in koledar sta namreč vezana na jezikovno nastavitev mobilne naprave.

- Podrobnosti zemljevida vplivajo na prikaz orientacijskih znakov in znamenitosti pri izrisu poti na zemljevid. Če je nastavitev aktivna, so dodatni elementi zemljevida vidni, v nasprotnem primeru pa zemljevid takšne elemente skrije.



Slika 3.17: Izgled uporabniškega vmesnika nastavitev.

Druga kategorija vsebuje osebne podatke. Zaenkrat aplikacija potrebuje le podatek o telesni teži v kilogramih, ki se uporablja za izračun porabljenih kalorij pri zajemu teka ali hoje.

Poglavje 4

Testiranje aplikacije

Predstavili bomo potek testiranja aplikacije v realnih primerih. Pogledali bomo, kaj smo testirali, kako smo testirali, in na koncu opisali rezultate testiranja.

4.1 Namen testiranja

S testiranjem smo želeli preveriti splošno delovanje aplikacije ter identificirati morebitne napake in nepravilnosti, ki bi se lahko pojavile med delovanjem. Testiranje smo zasnovali na realnem primeru uporabe aplikacije, v okviru katerega smo se osredotočili na tri glavna področja:

- testiranje funkcionalnosti;
- testiranje stabilnosti aplikacije ob prisotnosti prekinitev;
- testiranje odziva aplikacije v primeru napak med zajemom poti.

Za pridobljene podatke smo analizirali tudi uporabo datoteke GPX z orodji Google Earth Pro, Garmin BaseCamp, tekaško uro Garmin Forerunner 235 in aplikacijo Endomondo.

4.2 Testna mobilna naprava

Za testiranje smo uporabili pametni telefon Sony Xperia Z5, na katerem je nameščen operacijski sistem Android 7.0 (Nougat). Uporabljeni so bili tudi integrirani GPS, barometer in števec korakov.

4.3 Potek testiranja

Testiranje funkcionalnosti

Pri testiranju funkcionalnosti smo se osredotočili na pravilnost delovanja aplikacije. Testirali smo navigacijo med aktivnostmi in interakcije z ostalimi elementi aplikacije, proces zajema poti, shranjevanje pridobljenih podatkov v datoteko in njihov prikaz. Pri testiranju navigacije med aktivnostmi smo preverjali povezave v glavnem meniju in obvestilih, ki se lahko pojavijo med delovanjem. Pri zajemu poti smo preverjali pravilnost prikaza podatkov v realnem času in strukturo datoteke, v kateri so podatki shranjeni po opravljenem zajemu. Pri prikazu podatkov smo testirali delovanje filtrov, ki določajo prikaz vsebine in pravilnost prikaza podatkov, zemljevida in grafov v podrobnem pregledu posamezne zajete poti.

Testiranje stabilnosti med prekinitvami

Med delovanjem aplikacije se lahko pojavijo različne prekinitve, ki jih lahko sproži operacijski sistem, druge aplikacije, ki delujejo v ozadju, ali sam uporabnik. Osredotočili smo se na nekatere pogostejše prekinitve, kot so sprejemanje in pošiljanje sporočila, sprejem prihajajočega klica, prehod aplikacije v ozadje, delovanje v stanju pripravljenosti mobilne naprave in načinu za varčevanje z energijo (angl. Stamina Mode). Testiranje smo izvedli tako, da smo med delovanjem aplikacije na različnih mestih sprožili prekinitve. Iz druge naprave smo poslali sporočilo SMS in opravili klic. Za testiranje prehajanja aplikacije v ozadje smo med delovanjem pognali aplikacijo Gmail in

predvajalnik glasbe, nato pa v ospredje ponovno postavili testirano aplikacijo in preverili delovanje. Prav tako smo med delovanjem postavili mobilno napravo v stanje pripravljenosti in v nastavitvah spremenili porabo baterije.

Testiranje odzivnosti v primeru napake

Pri testiranju odziva aplikacije na napake smo se še posebej osredotočili na izpad signala GPS in prekinitev internetne povezave. Določene funkcionalnosti aplikacije so odvisne od internetne povezave, zato je ustrezna odzivnost v primeru takšne napake ključna, saj na določenih predelih kljub dobri pokritosti mobilnih omrežij prihaja do izgube povezave. Za GPS predstavljajo največjo oviro predori, kjer je izguba satelitskega signala neizogibna. Testiranje smo izvedli tako, da smo med zajemom poti simulirali izpad internetne povezave s prekinitvijo podatkovnega prometa. Izpad signala GPS pa smo testirali z vožnjo skozi predor Golovec (622 m) na ljubljanskem avtocestnem obroču. Testirali smo tudi obvestila, ki jih aplikacija posreduje med izvajanjem ostalih aktivnosti. Testirali smo primere, kot so poskus zajema poti brez omogočene uporabe lokacije, zapis značke NFC brez omogočene uporabe NFC storitev in izhod iz aktivnosti za zajem poti brez shranjevanja.

4.4 Rezultati in ugotovitve testiranja

Aplikacija je večino testiranj opravila uspešno. Pri testiranju funkcionalnosti nismo odkrili večjih nepravilnosti. Med navigacijo po glavnem meniju smo ugotovili, da se določeni elementi uporabniškega vmesnika večkrat naložijo ob zaporedni izbiri iste opcije menija. Težavo smo odpravili z dodatnim preverjanjem pred zamenjavo fragmenta, ki prikazuje vsebino. Obvestila, ki se pojavijo ob določenih interakcijah, delujejo brezhibno. Prikaz podatkov med zajemom poti deluje brezhibno. Zajeti podatki se pravilno shranjujejo v datoteko po opravljenem zajemu. Pri zapisu datoteke v pomnilnik naprave smo ugotovili nepravilnost, ki sicer ni povezana s samo programsko kodo. Shranjene datoteke namreč niso bile vidne ob priklopu mobilne naprave na

računalnik z uporabo kabla USB. Težavo je povzročal protokol MTP (angl. Media Transfer Protocol), ki ne zazna programsko ustvarjenih datotek [2]. Težavo smo odpravili z uporabo objekta tipa `MediaScannerConnection`, ki takoj po zapisu datoteke izvede ponastavitev razpoložljivih datotek. Filtriranje prikaza vsebine in podrobni pregled posamezne poti deluje brez težav.

Pri testiranju stabilnosti med prekinitvami nismo odkrili nobenih nepravilnosti pri prehodu aplikacije v ozadje ali pri prejemanju sporočil SMS in klicev. Aplikacija nemoteno deluje, tudi ko je naprava v stanju pripravljenosti. Na delovanje je vplival le način za varčevanje energije, ki sicer ne ovira nobene funkcionalnosti aplikacije, vendar nekoliko upočasni prikaz gradnikov uporabniškega vmesnika.

V primeru napake med zajemom poti aplikacija deluje nemoteno. Izpad internetne povezave vpliva na natančnost izračuna nadmorske višine in prikaz temperature. Oba podatka sta namreč odvisna od podatkov vmesnika OpenWeatherMap. V takem primeru bo aplikacija nadaljevala z uporabo zadnjih pridobljenih podatkov in ob naslednjem intervalu ponovno poskušala pridobiti sveže podatke. Tudi ob izpadu signala GPS se delovanje aplikacije ne prekine. Zajem lokacije se nadaljuje takoj, ko naprava ponovno sprejme podatke od satelitov. Pri vožnji skozi predor se izračuna razdalja med zadnjo lokacijo pred začetkom predora in prvo lokacijo izven predora. Napaka pri izračunu razdalje je odvisna od oblike predora (predor z ovinkom ima večjo napako).

Tudi zapis značke deluje pravilno. Če značka ne podpira sporočila NDEF, se na uporabniškem vmesniku pojavi obvestilo, ki nakazuje na nezdružljivost pasivne naprave NFC.

Poglavje 5

Sklepne ugotovitve

V diplomski nalogi je predstavljen prototip mobilne aplikacije za spremljanje poti na osnovi podatkov, pridobljenih z uporabo tehnologije GPS, vgrajenih senzorjev in zunanjih vmesnikov (Google Map APIs, OpenWeatherMap API). Aplikacija med izvajanjem poti beleži podatke v realnem času ter jih po končanem spremljanju zapiše v datoteko. Na osnovi shranjenih podatkov prikaže zgodovino gibanja in omogoča izvedbo analize posameznih poti. Poleg razvoja aplikacije smo predstavili tudi uporabljene tehnologije, implementacijo algoritmov ter postopek in rezultate testiranja.

Poseben izziv pri razvoju aplikacije je predstavljal podatek o nadmorski višini posameznih lokacij, ki jih posreduje GPS. Njihova nadmorska višina je namreč določena kot odmik od elipsoide v referenčnem modelu WGS-84 (angl. WGS-84 Reference Ellipsoid). Ugotovili smo, da je izračun realne nadmorske višine na osnovi odmika procesorsko preveč zahteven za izvedbo na mobilni napravi v realnem času. Nadmorsko višino bi lahko pridobili z uporabo enega od spletnih vmesnikov, na primer Google Maps Elevation, vendar bi s tem bistveno povečali odvisnost aplikacije od internetne povezave, čemur smo se poskušali izogniti. Zato smo za izračun uporabili hipsometrično enačbo, ki deluje na osnovi temperature in zračnega pritiska. Na ta način smo zagotovili izračun nadmorske višine z zadovoljivo natančnostjo in minimizirali

odvisnost aplikacije od internetne povezave.

Pozornost smo namenili tudi optimizaciji datoteke, v kateri so shranjeni podatki poti. Ugotovili smo, da največ prostora zavzame zapis geografskih koordinat. Koordinata je definirana z zemljepisno širino in dolžino. Gre za dve predznačeni decimalni števili, kjer za zapis vsakega potrebujemo povprečno trinajst znakov. Če upoštevamo še dejstvo, da so vrednosti zapisane v obliki parov ključ/vrednost (angl. Key/Value), se število znakov še poveča. Zapis geografskih koordinat smo zato kodirali z uporabo kompresijskega algoritma in s tem učinkovito zmanjšali povprečno velikost datoteke.

Pomemben del aplikacije predstavlja tudi uporabniški vmesnik. Aplikacija lahko brez težav deluje v ozadju, vendar med spremljanjem poti omogoča prikaz opravljene razdalje, časa trajanja, nadmorske višine, smeri potovanja, temperature ter število opravljenih korakov in tempa pri teku oziroma hoji v realnem času. Prav tako smo z ustrezno izbiro gradnikov poskušali prikazati rezultate analize poti. Uporabniški vmesnik temelji na principu "Material Design". Gre za način izdelave vmesnika, ki omogoča enotno uporabniško izkušnjo na različnih platformah in velikostih mobilnih naprav. Omogoča tudi uporabo naprednih gradnikov, ki jih starejše različice Androida ne podpirajo. V uporabniški vmesnik so vključeni tudi zemljevid Google Map, ki izriše opravljeno pot, ter stolpni in linearni grafi, ki so izrisani z uporabo zunanje knjižnice MPAndroidChart.

Razvita aplikacija predstavlja uspešno zasnovan prototip, ki bi ga potencialno lahko prilagodili za različne primere uporabe. Obstaja več možnosti za nadaljni razvoj, optimizacijo in dodajanje novih funkcionalnosti. Med spremljanjem teka in hoje bi lahko z uporabo zunanjega merilca spremljali srčni utrip in na njegovi osnovi natančneje izračunali število porabljenih kalorij. Trenutno se ta podatek po končanem spremljanju približno izračuna glede na porabljeni čas, telesno težo in vrednost MET (angl. Metabolic equivalent).

S tem bi aplikacijo lahko prilagodili za spremljanje različnih fizičnih aktivnosti. Druga možnost nadgradnje je usmerjena v področje avtomobilizma. Spremljanje vožnje bi lahko nadgradili z uporabo adapterja OBD (angl. On-board diagnostics), ki omogoča pridobitev podatkov direktno iz računalnika v vozilu.

Literatura

[1] Android dokumentacija. Dostopno na: https://developer.android.com/index.html. [Dostopano: 21. 11. 2017].

[2] Baschi. Folder added in android not visible via USB. Dostopno na: https://stackoverflow.com/questions/13507789/folder-added-in-android-not-visible-via-usb, 2013. [Dostopano 20. 12. 2017].

[3] Isaiah David. How Do Piezoresistive Pressure Sensors Work. Dostopno na: https://sciencing.com/piezoresistive-pressure-sensors-work-4609318.html, 2017. [Dostopano 24. 12. 2017].

[4] Android Developers. Android 4.4: Step sensors. Dostopno na: https://www.youtube.com/watch?v=yv9jskPvLUc, 2014. [Dostopano: 30. 12. 2017].

[5] Dokumentacija za kodiranje koordinat. Dostopno na: https://developers.google.com/maps/documentation/utilities/polylinealgorithm. [Dostopano: 25. 8. 2017].

[6] Enačba za izračun nadmorske višine. Dostopno na: http://keisan.casio.com/exec/system/1224585971. [Dostopano: 20. 9. 2017].

[7] Enačba za izračun naklona. Dostopno na: https://www.archtoolbox.com/measurements/geometry/slope.html. [Dostopano: 18. 1. 2018].

[8] Enačba za izračun zračnega pritiska. Dostopno na: http://keisan.casio.com/exec/system/1224579725. [Dostopano: 20. 9. 2017].

[9] Kaj je gps. Dostopno na: https://www.gps.gov/systems/gps/. [Dostopano: 29. 12. 2017].

[10] Knjižnica gpxparser. Dostopno na: http://gpxparser.alternativevision.ro/pages/index.html. [Dostopano: 25. 8. 2017].

[11] Knjižnica mpandroidchart. Dostopno na: https://github.com/PhilJay/MPAndroidChart. [Dostopano: 25. 8. 2017].

[12] Met vrednosti. Dostopno na: https://sites.google.com/site/compendiumofphysicalactivities/Activity-Categories. [Dostopano: 20. 1. 2018].

[13] Napake pri določanju lokacije. Dostopno na: http://gisgeography.com/gps-accuracy-hdop-pdop-gdop-multipath/. [Dostopano: 30. 12. 2017].

[14] Natančnost gps. Dostopno na: https://www.gps.gov/systems/gps/performance/accuracy/. [Dostopano: 30. 12. 2017].

[15] Nfc forum. Dostopno na: https://nfc-forum.org/our-work/specifications-and-application-documents/. [Dostopano: 1. 1. 2018].

[16] The Editors of Encyclopædia Britannica. Laps rate. Dostopno na: https://www.britannica.com/science/lapse-rate, 2009. [Dostopano: 20. 1. 2018].

[17] Openweathermap. Dostopno na: https://openweathermap.org/api. [Dostopano: 1. 1. 2018].

[18] NFC Data Exchange Format (NDEF). Dostopno na: http://sweet.ua.pt/andre.zuquete/Aulas/IRFID/11-12/docs/NFC%20Data%20Exchange%20Format%20(NDEF).pdf, 2006. [Dostopano: 15. 1. 2018].

[19] Princip trilateracije. Dostopno na: http://gisgeography.com/trilateration-triangulation-gps/. [Dostopano: 30. 12. 2017].

[20] Emil Protalinski. Google releases Android Studio 1.0, the first stable version of its IDE. Dostopno na: https://venturebeat.com/2014/12/08/google-releases-android-studio-1-0-the-first-stable-version-of-its-ide/, 2014. [Dostopano 24. 12. 2017].

[21] Tehnologija mems. Dostopno na: http://www.memsnet.org/mems/what_is.html. [Dostopano: 1. 1. 2018].

[22] Tržni delež android platform. Dostopno na: https://developer.android.com/about/dashboards/index.html. [Dostopano: 6. 1. 2018].

[23] Vrste signalov gps satelitov. Dostopno na: https://www.gps.gov/systems/gps/modernization/civilsignals/#L5. [Dostopano: 29. 12. 2017].

[24] Ningchuan Xiao. *GIS Algorithms*. SAGE, 2016.